BR-322501

AD-A278 422

(1)

68pp

DTIC
ELECTE
APR 25 1994
S F D

This document has been approved for public release and sale; its distribution is unlimited

94-12313

DTIC QUALITY INSPECTED 3

94 4 22 002

United Kingdom Atomic Energy Authority

HARWELL

Fortran subroutines for finding polynomial zeros

K. Madsen and J.K. Reid
Computer Science and Systems Division
AERE Harwell, Oxfordshire
February 1975

PRICE £2·00 net from H.M. Stationery Office C13

A.E.R.E. R.7986

FORTRAN SUBROUTINES FOR FINDING POLYNOMIAL ZEROS

by

K. Madsen* and J.K. Reid

ABSTRACT

In this report we present subroutines for finding all the roots of a polynomial and bounds on their errors. To find the zeros we use the algorithm of Madsen (1973) and to find error bounds we use the work of Peters and Wilkinson (1971) with some significant modifications. Both the real and complex cases are treated.

* Institute for Numerical Analysis, Technical University of Denmark, 2800 Lyngby, Denmark.

Computer Science and Systems Division,
A.E.R.E. Harwell, Didcot,
Oxford, England.

February, 1975

HL.75/1172 (C.13)

Such bounds are particularly unrealistic in such a case as

CONTENTS

ISBN-0-70-580215-9

1. Introduction

In this report we consider the problem of finding all the zeros of the polynomial

$$f(z) = a_0 + a_1z + \ldots + a_nz^n \qquad (1.1)$$

and estimating error bounds for them. To find the set of zeros we use the algorithm of Madsen (1973), which we have found to compare favourab with other algorithms. For error estimation we apply Rouché's theorem as recommended by Peters and Wilkinson (1971), but with some difference in detail. Both of these algorithms are a little simpler in the case where the polynomial has complex coefficients, so we describe the algorithms for this case in sections 2 and 3 and the modifications for t real case in section 4.

We believe that our code is in accord with the ANSI standard, havir checked it with Bell Telephone Laboratories' Fortran verifier (Ryder, 1973) and run our test programs with array subscript checking. We describe this code in section 5 and in an appendix give specification sheets, and listings (produced by the Bell Laboratories' verifier, so th include cross references). For the Harwell subroutine library we have made a small number of changes in order to shorten the argument lists, at the expense of a departure from ANSI standard. We have decided aga including a single-length version in the library because the IBM 370/16 has a very short single-length word (6 hexadecimal digits, so that numbers just greater than unity are held to about 1 part in 10^6) but ha double-length hardware which executes very little slower than the singl length hardware. Some results obtained with the library subroutines, together with comparisons with other algorithms are given in section 6. Our code (listed in section 7) contains comments suitable for machine processing which detail the changes needed for the single-length and

Harwell library versions.

We would like to acknowledge the help of M.J.D. Powell in carefully checking a draft of this report and making several valuable suggestions.

2. Finding the roots in the complex case

We use the algorithm of Madsen (1973) to find the root of minimal or near-minimal modulus and then use forward deflation to construct a polynomial of degree n-1 whose roots are the remaining roots of the original polynomial. The process is repeated until approximations to all the roots have been found. Wilkinson (1963) has shown that forward deflation is stable provided a large root is not accepted before a much smaller one. Our algorithm does not guarantee that the moduli of the roots are strictly increasing but our experience has always been that they are found in roughly increasing order. A version with the composite deflation of Peters and Wilkinson (1971), which should be stable no matter in which order the zeros are found, was tried but it did not give more accurate results. In any case it is not clear how to apply the composite deflation when two complex conjugate roots of a real polynomial have been found so that deflation by a real quadratic factor is wanted.

It remains necessary to describe Madsen's (1973) algorithm in detail and we will consider its application to the original polynomial. The general strategy of the algorithm is that, given an iterate z_k, a tentative step dz_k is found and the next iterate z_{k+1} is taken at the best point (in the sense of $|f(z)|$) encountered in a short search of values on the line through z_k and z_k+dz_k. Because the search may sometimes yield no better value than that at z_k we may sometimes have $z_{k+1}=z_k$ and in this case ensure that the next tentative step is shorter and in a different direction. The inclusion of searches ensures rapid convergence to multiple roots and reliable convergence when difficulties are

ATTRIBUTES: Col 1: C if in COMMON

encountered. Such a search is, however, wasteful if we are so near a simple root that Newton's iteration is reliable and fast. We have therefore devised a test (given by inequality (2.7)) which normally ensures this. While the algorithm performs searches we say it is in stage 1; otherwise it is in stage 2, performing straightforward Newton iteration. It begins in stage 1, which we now describe.

The tentative step dz_k is found with the help of stored values of z_k, $f(z_k)$, $f'(z_k)$, z_{k-1} and the previous tentative step dz_{k-1}. If the last iteration was successful ($z_k \neq z_{k-1}$) then the Newton correction

$$n_k = -f(z_k)/f'(z_k) \qquad (2.2)$$

is calculated and the next tentative step is taken as

$$dz_k = \begin{cases} n_k & \text{if } |n_k| \leq 3|z_k - z_{k-1}| \qquad (2.3a) \\ 3|z_k - z_{k-1}|e^{i\theta}\, n_k/|n_k| & \text{otherwise} \qquad (2.3b) \end{cases}$$

where θ is chosen (rather arbitrarily) as arctan(3/4). If the last step was unsuccessful ($z_k = z_{k-1}$) then we take the tentative step to be

$$dz_k = -\tfrac{1}{2}\, e^{i\theta}\, dz_{k-1}. \qquad (2.3c)$$

After a successful iteration we normally expect to want to take a Newton step (2.3a), but we include the alternative (2.3b) because accidently coming near to a stationary point of f(z) is likely to make the Newton step ridiculously large. We include a change of direction in (2.3b) because if a saddle point is being approached the direction n_k may be a worse search direction than almost any other. After an unsuccessful iteration we want to change the search direction to one likely to be successful and reduce the step size; this leads to formula (2.3c). Its repeated use is sure to yield a descent direction.

The algorithm used by PA06BD to find the roots has already been

Once the tentative step has been found we test the inequality

$$|f(z_k+dz_k)| < |f(z_k)| \ . \qquad (2.4)$$

If this is satisfied then we calculate the numbers

$$|f(z_k+p\ dz_k)|,\ p=1,2,\ldots,n \qquad (2.5)$$

continuing for as long as these are strictly decreasing. If inequality (2.4) does not hold th.n we calculate the numbers

$$|f(z_k + dz_k/2^p)|,\ p=0,1,2,\ |f(z_k + \tfrac{1}{4}e^{i\theta}\ dz_k)| \qquad (2.6)$$

again continuing until the sequence ceases to decrease. In all cases we take z_{k+1} to be the best point found. Note that if there is a true multiple zero of multiplicity m or if we are a fair distance from a cluster of m zeros then z_k+mn_k will be a very good estimate of the solution and will be found by our search. In fact we get quadratic convergence to a multiple zero. Note also that when an iteration fails following a search to the end of sequence (2.6), the choice (2.3c) leads to a step likely to be in a direction of decreasing $|f(z)|$.

To complete our description of stage 1 we need to specify starting iterates. We take these to be

$$\left.\begin{aligned} z_0 &= 0 \\ dz_0 &= \begin{cases} -f(0)/f'(0) & \text{if } f'(0)=0 \\ 1 & \text{otherwise} \end{cases} \\ z_1 &= \tfrac{1}{2} \min_{k>o} \left(\left|\frac{a_0}{a_k}\right|^{1/k} \right) \frac{dz_0}{|dz_0|} \ . \end{aligned}\right\} \qquad (2.7)$$

The iteration really starts from z_1 but we have to include z_0 and dz_0 because they are needed for choosing the tentative step dz_1. This choice of z_1 is used because its modulus is certainly less than that of any root of f(z) and it is in the direction of steepest descent of $|f(z)|$ from the

origin. It is therefore likely that we will converge to a root of near-minimal modulus.

We do not make our main test for switching to stage 2 (straightforward Newton iteration) until a stage 1 search has led to the choice $z_{k+1}=z_k+dz_k$. This is obviously sensible and has the added virtue that we should (correctly) avoid the switch when converging to a multiple root. The test itself is based on the Kantorowitz theorem (see, for example, Ostrowski (1966)) which states that if K_0 is the circle with centre z_k+n_k and radius $|n_k|$ where n_k is the Newton step (2.2) then the conditions

$$\left.\begin{aligned} &f(z_k)\ f'(z_k) \neq 0 \\ &2|f(z_k)| \max_{z\varepsilon K_0} |f''(z)| \leq |f'(z_k)|^2 \end{aligned}\right\} \qquad (2.8)$$

ensure the convergence of Newton's iteration starting from z_k. This leads us to test the inequality

$$2|f(z_k)||f'(z_{k-1}) - f'(z_k)| \leq |f'(z_k)|^2\ |z_{k-1}-z_k| \ . \qquad (2.9)$$

Of course this is not equivalent to test (2.8) because we have replaced $\max_{z\varepsilon K_0} |f''(z)|$ by a rather crude difference approximation but we nevertheless expect it usually to predict correctly that straightforward Newton iteration will be satisfactory. We check inequality (2.9) at every step in stage 2 and switch back to stage 1 if it is violated. We also check the inequality (2.4) and if this is violated return to stage 1 beginning by modifying the tentative step with formula (2.3c) as in stage 1.

We complete this section by describing our convergence criterion. We terminate if a stage 1 search or a stage 2 iteration leads to a new iterate z_{k+1} different from z_k and yet such that the inequality

$$|z_{k+1}-z_k| < \varepsilon|z_{k+1}| \tag{2.10}$$

holds where ε is the largest number such that to machine accuracy $1+\varepsilon = 1$. We also terminate if the condition

$$|f(z_{k+1})| = |f(z_k)| \le 16n|a_0|\varepsilon \tag{2.11}$$

holds. The expression $16n|a_0|\varepsilon$ is a generous overestimate of the final roundoff made in calculating $f(z)$ at the root of smallest modulus and we expect that such accuracy will be attainable. The normal convergence pattern is that $|f(z_k)|$ decreases until well below $16n|a_0|\varepsilon$ and then roundoff errors cause a new iterate $z_{k+1}=z_k$ to be taken so that (2.11) is satisfied. If such accuracy is unattainable then the step will decrease steadily because of the application of (2.3c) until (2.10) is satisfied. This combination of convergence criteria means that we are certain to obtain a good solution and almost certain to obtain the best possible. Furthermore this result is usually obtained with only one more iteration than is necessary to get this good accuracy.

3. Error estimation

We seek to estimate error bounds for all the roots produced by the algorithm of the previous section. We base our algorithm on that of Peters and Wilkinson (1971) and will follow their notation. The most significant difference is that they look for non-overlapping discs each of which contains precisely the same number of exact and approximate roots, whereas we allow overlapping discs. This is because it can happen that one root is well determined although it is inside the best disc obtainable for another (ill-determined) root. We therefore look for a separate disc for each root and take its centre to be at the calculated root itself. This also allows a very simple form of output to the user since

all that is required is a radius for each root.

We suppose that we have approximate roots α_i, $i=1,2,\dots,n$. Then the polynomial

$$P(z) = a_n \prod_{i=1}^{n} (z-\alpha_i) \tag{3.1}$$

should agree with the original polynomial (1.1), but because the roots are not exact there will be an error

$$Q(z) = P(z) - f(z). \tag{3.2}$$

Now Rouché's theorem states that if $P(z)$ and $Q(z)$ are analytic functions in and on the closed curve C and the inequality

$$|Q(z)| < |P(z)| \tag{3.3}$$

holds on C then $P(z)$ and $P(z)-Q(z)$ have the same number of zeros inside C. We apply this here by looking for circles with centres α_i, $i=1,2,\dots,n$ on which condition (3.3) holds. The following theorem shows that there is no need to worry about the overlapping of some of these discs.

Theorem 1 If condition (3.3) holds on each of the circles centre α_i, radius r_i, $i=1,2,\dots,n$, then the roots ϕ_i of $f(z)$ may be ordered so that

$$|\alpha_i-\phi_i| \le r_i, \quad i=1,2,\dots,n . \tag{3.4}$$

Proof Regard the perimeters of the circles as dividing the plane into a set of non-overlapping regions R_i, each of which is the intersection of a subset of the set of discs that the circles enclose and their complements. A simple case is illustrated in Figure 1. Let R_{k_i} be the region containing α_i, $i=1,2,\dots,n$. Note that a region may contain more than one α_i so that there may be coincidences among the k_i (e.g. $k_1=k_2=3$ in Figure 1). The set of regions R_{k_i}, $i=1,2,\dots,n$ together contain all the

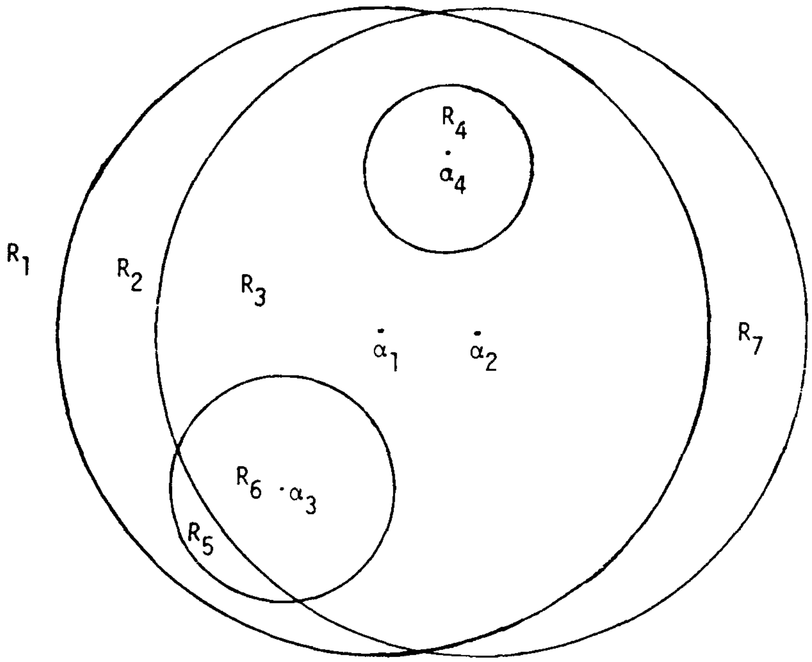

Figure 1 Four circles centre α_i dividing plane into seven regions R_i

roots α_i of $P(z)$ and each R_{k_i} has a boundary on which condition (3.3) holds so contains exactly the same number of roots of $f(z)$ as of $P(z)$. Therefore the regions R_{k_i} $(i=1,2,\ldots,n)$ contain all the roots of $f(z)$ and these may be ordered so that R_{k_i} contains ϕ_i $(i=1,2,\ldots,n)$. The result now follows since each R_{k_i} is contained in the disc centre α_i radius r_i. ■

Being able to use overlapping discs leads to simplifications in coding and sometimes to much better error bounds. This improvement is illustrated by the example shown in Figure 1, where α_1 and α_2 are quite ill-conditioned but α_3 and α_4 are not. The procedure of Peters and Wilkinson would have forced us to regard all four α_i as a cluster and to enclose them in a single disc. This would have made little difference to the bounds for α_1 and α_2 but would have significantly worsened the bounds

for α_3 and α_4. Another illustration of this point is given in §6 with an example using our code.

We now suppose that the error polynomial is

$$Q(z) = \sum_{i=0}^{n} \varepsilon_i z^i \tag{3.5}$$

and describe how to find a suitable circle for a typical root. For simplicity of notation let us imagine that the roots are reordered so that the one for which we are seeking a circle is α_1 and $|\alpha_i-\alpha_1|$ increases monotonically with i. Rouché's condition (3.3) is satisfied on a circle centre α_1 and radius r if the inequality

$$\sum_{i=0}^{n} |\varepsilon_i| \ (r+|\alpha_1|)^i < \left|a_n \prod_{i=1}^{n} \left(|\alpha_i-\alpha_1| - r\right)\right| \tag{3.6}$$

holds. If m is such that

$$|\alpha_m-\alpha_1| < r < |\alpha_{m-1}-\alpha_1| \tag{3.7}$$

(we ignore the case $r = |\alpha_i-\alpha_1|$ for some i because clearly inequality (3.6) cannot be satisfied in this case) then we may rewrite inequality (3.6) in the form

$$k(r) < \prod_{i=1}^{m} \left(r - |\alpha_i-\alpha_1|\right) \tag{3.8}$$

where k(r) is given by the equation

$$k(r) = \frac{\sum_{i=0}^{n} |\varepsilon_i| \ (r+|\alpha_1|)^i}{|a_n| \prod_{i=m+1}^{n} \left(|\alpha_i-\alpha_1|-r\right)} . \tag{3.9}$$

Peters and Wilkinson (1971) solve the equation

$$(1.1)\ k(0) = \left(r_1 - \max_{1\le i\le m} |\alpha_i-\alpha_1|\right)^m \tag{3.10}$$

and then check whether r_1 satisfies inequalities (3.7) and (3.8). Almost always inequality (3.8) holds because

(i) the right hand side of (3.10) underestimates the corresponding expression in (3.8),

(ii) the safety factor (1.1) has been introduced

and (iii) $k(r)$ is nearly constant in the usual case where the roots α_i, $i=1,2,\ldots,m$ are well separated from the rest.

They give no recommendation for dealing with the case where r_1 does not satisfy (3.8), but presumably intend that an iteration should be set up with 0 and r_1 in (3.10) being replaced by r_j and r_{j+1}. This yields a montonically increasing sequence, so that we must eventually either satisfy inequality (3.8) or break the right-hand inequality (3.7) necessitating a new start with a greater value for m. Actually Peters and Wilkinson take the centre of their circle to be $\bar{\alpha} = \sum_{i=1}^{m} \alpha_i/m$ rather than α_1 and this means that the right-hand side of (3.10) is quite a good estimate of the right-hand side of (3.8) so that the procedure gives a realistic radius r without very much computation. Unfortunately this is not the case with α_1 in use, except when m=1 and we have sometimes obtained solutions such that the right-hand side of (3.8) exceeds the left by factors as big as 1,000. We have therefore decided to use an iteration based directly on (3.8). Given an iterate r_j we seek a new iterate such that

$$(1.05)\, k(r_j) < \prod_{i=1}^{m} (r_{j+1} - |\alpha_i-\alpha_1|) < (1.1)\, k(r_j). \qquad (3.11)$$

In view of inequality (3.7) it seems sensible to start with $r_0 = |\alpha_m-\alpha_1|$ rather than Wilkinson and Peters' $r_0=0$. To solve (3.11) we use the method of bisection because m is usually small so that function evaluations are cheap, no great accuracy is required in view of the slack in (3.11),

S (double-length standard version only) is a REAL work array of

and suitable initial upper and lower bounds are available in r_j and the Peters and Wilkinson overestimate obtained from the analogue of equation (3.10). Typically between four and seven iterates are required.

Peters and Wilkinson suggest that r_{j+1} should be checked directly in (3.8) but a simple test often avoids any need for this. If the inequality

$$\left(\frac{r_{j+1} + |\alpha_1|}{r_j + |\alpha_1|} \middle/ \frac{|\alpha_{m+1} - \alpha_1| - r_{j+1}}{|\alpha_{m+1} - \alpha_1| - r_j}\right)^n \leq 1.05 \qquad (3.12)$$

holds then it follows from the definition (3.9) that the inequality

$$k(r_{j+1}) \leq 1.05\ k(r_j) \qquad (3.13)$$

also holds. From this last inequality and the first inequality (3.11) we deduce that r_{j+1} satisfies inequality (3.8).

Rounding errors occur when the coefficients of P(z) are calculated, but can be reduced by multiplying out the factors $(z-\alpha_i)$ in order of increasing $|\alpha_i|$. We therefore use the same order as that in which they were calculated. If bounds e_i on the errors were available we could work with the error polynomial $\sum_{i=o}^{n} (|\varepsilon_i|+e_i)z^i$ in place of the polynomial (3.5) and obtain strict bounds on the errors in the calculated roots. We hoped to use the running error analysis of Peters and Wilkinson for this purpose, but unfortunately found that it sometimes gave gross overestimates, as explained in the last paragraph of this section. We therefore have ignored this source of error, relying on such errors being reflected in larger coefficients ε_i in the polynomial Q(z). It should be noted that the sequence of constructed polynomials $\prod_{i=1}^{r} (z-\alpha_i)$, r=1,2,...,n is not identical with the sequence of deflated polynomials used when finding the roots because we perform the multiplication in the same order as that in which the roots were found. Therefore errors produced by the

multiplication are likely to show in enlarged coefficients ε_i. Not bounding these errors means that our bounds are not strict bounds, but they are more realistic. In none of our tests did the actual errors exceed the estimated bounds.

It is straightforward to allow for the effects of uncertainties in the original coefficients by working with the error polynomial $\sum_{i=0}^{n} (|e_i|+b_i)$ where b_i, $i=0,1,\ldots,n$, are bounds on these errors. This we do in our subroutine.

We complete this section by explaining why the running error analysis of Peters and Wilkinson for the coefficients of $\prod_{i=1}^{n} (z-\alpha_i)$ sometimes gives very pessimistic results. If the computed product $\prod_{i=1}^{r} (z-\alpha_i)$ is $\sum_{i=0}^{r} b_i^{(r)} z^i$ then they generate coefficients $f_i^{(r)}$ from the recurrences.

$$
\left.
\begin{aligned}
f_0^{(1)} &= f_1^{(1)} = 0 \\
f_0^{(r+1)} &= |\alpha_{r+1}| f_0^{(r)} + |b_0^{(r+1)}| \\
f_i^{(r+1)} &= f_{i-1}^{(r)} + |\alpha_{r+1}| f_i^{(r)} + |\alpha_{r+1}||b_i^{(r)}| + |b_i^{(r+1)}|, \\
& \qquad\qquad i=1,2,\ldots,r \\
f_{-1}^{(r+1)} &= f_{r+1}^{(r+1)} = 0
\end{aligned}
\right\}
\quad r=1,2,\ldots,n-1 \tag{3.14}
$$

to yield bounds $2^{-t} f_i^{(n)}$ for the coefficients of $P(z)$ if floating-point computation with t binary places is used. This can sometimes lead to very pessimistic bounds since the recurrence (3.14) fails to distinguish properly between $\Pi(z-\alpha_i)$ and $\Pi(z+|\alpha_i|)$. In fact the recurrence (3.14) is obtained by bounding the corresponding recurrence

$$e_i^{(r+1)} = e_{i-1}^{(r)} - \alpha_{r+1} e_i^{(r)} - \varepsilon_1 \alpha_{r+1} b_i^{(r)} - \varepsilon_2 b_i^{(r+1)} . \tag{3.15}$$

NP1 (standard versions only) is an INTEGER which must be set to n+1.

Such bounds are particularly unrealistic in such a case as $f(z) = z^n-1$ since the numbers $f^{(r)}$ grow with r in much the same way as the recurrence for generating the coefficients of $(z+1)^n$ and these coefficients are ${}^nCr, r=0,1,\ldots,n$. In fact the recurrence (3.14) has some added terms when compared with the recurrence for the coefficients of $(z+1)^n$.

4. Changes to the algorithms in the real case

The only change made to our root-finding algorithm in the case where f(z) has real coefficients is that once a complex root has been found it is either perturbed into a real root or its conjugate is taken as another root. This ensures that each deflated polynomial is also real and that work is saved for genuinely complex roots. Once a complex root $\alpha_k=x_k+iy_k$ has been found we evaluate $f(x_k)$ and use Peters and Wilkinson's (1971) running error analysis to bound the roundoff error made in this evaluation. The recurrence used for evaluating $f(x_k)$ is given by the equations

$$\left.\begin{aligned} s_n &= a_n \\ s_i &= x_k s_{i+1}+a_i, \quad i=n-1,\ldots,0 \\ f(x_k) &= s_0 \end{aligned}\right\} \qquad (4.1)$$

and the corresponding running error analysis is given by the equations

$$\left.\begin{aligned} g_n &= 0 \\ g_i &= |x_k|(g_{i+1}+|s_{i+1}|) + |s_i|, \; i=n-1,\ldots,0 \end{aligned}\right\} \qquad (4.2)$$

to yield the bound εg_0 for the error in $f(x_k)$ where ε is the relative floating-point accuracy. If the inequality

$$|s_0| < 2\varepsilon g_0 + |f(\alpha_k)| \qquad (4.3)$$

holds (where $f(\alpha_k)$ is the computed value at α_k), we take x_k to be a real root and deflate with it. Otherwise we deflate with the complex conjugate pair $x_k \pm iy_k$. It is very important that the case where we have a simple

real root well separated from the other roots should not be taken as a complex pair because a double deflation in such a case would be disastrous. We therefore consider this case and suppose that ϕ_k is such a real root and α_k is the (complex) approximation to it found by the algorithm. x_k (real part of α_k) is a better approximation to ϕ_k in the sense that the inequality

$$|\phi_k - x_k| \le |\phi_k - \alpha_k| \tag{4.4}$$

holds since ϕ_k is real. Further the exact value f(z) behaves like a constant times $(z-\phi_k)$ if z is near ϕ_k. It is therefore reasonable to expect that the inequality

$$|f(x_k)| \le |f(\alpha_k)| \tag{4.5}$$

holds between the corresponding exact values. We therefore expect x_k to be at least as good a root as α_k so that it is virtually certain that inequality (4.3) will hold, since εg_0 is a rigorous upper bound on the round-off error in computing $f(x_k)$ and may also be taken (slightly incorrectly) as a bound on the error in computing $|f(\alpha_k)|$.

A disadvantage of the test (4.3) is that we may decide we have a real root when a complex conjugate pair is the true position. We feel that this disadvantage is quite slight because it is reasonable to regard any point for which inequality (4.3) holds as a good approximate root.

The only change to the error bounding algorithm is that once a bound for a complex root has been found, this bound is also used for its complex conjugate.

5. Description of Fortran subroutines

In this section we describe the Fortran subroutines themselves. It is a short section because we have included rather full comments in the code itself, since we believe that this provides the most convenient form of documentation. Specification sheets and listings of the code are given in section 7. The specification sheets document all three versions (double- and singe-length standard Fortran and double-length IBM Fortran). The listings are of the standard Fortran double-length versions and contain specially coded comments detailing changes needed for the other versions. We have been using a simple preprocessor (written in standard Fortran) to convert from one version to another. Where a statement differs between versions we preceded the genuine (double-length standard) statement by the alternative version or versions modified by the insertion of C in column 1, / in column 72, and the letter I or S (for IBM version or single-length version) in column 71.

The listings used are those produced by the Bell Telephone Laboratories' Fortran verifier because they include cross-references which are very helpful when reading the code. They consist of lists of all the identifiers and labels in lexographical order together with the statement numbers of all references to them and the following coded information for each identifier:

```
TYPE:       Col 1:  E if explicitly typed
            Col 2:  I for integer
                    R for real
                    D for double precision
                    C for complex
                    L for logical
                    H for Hollerith

USE:                FA for arithmetic statement function argument
                    FN for function
                    E  for external subroutine or function
                    GT for assigned "go to" variable
                    IF for intrinsic function
                    SN for subroutine
                    V  for variable
```

ATTRIBUTES:	Col 1:	C if in COMMON
	Col 2:	E if an EQUIVALENCE
	Col 3:	A if a dummy argument
	Col 4:	S if value set by program unit
	Col 5:	S if scalar A if array
	Col 6:	if array then number of dimensions

Because double-length complex facilities are not available in standard Fortran we do not use any complex variables, although the arguments A,R and E may be regarded as COMPLEX*16 (or COMPLEX for the single-length version) on the IBM 360/370 series. All complex variables are written as arrays of length two and all complex arrays are written as 2-dimensional arrays whose first dimension is two. This leads to the code being almost as easy to read as if the complex facilities were used. Complex division is more complicated than can be written conveniently as in-line code so we have taken this out of line to the subroutines PA06ED/7ED.

We begin by describing the versions for complex polynomials, that is PA06AD/BD/CD/DD/ED.

The main call is to a short routine PA06AD which calls PA06BD to find the roots and PA06CD to find error bounds. PA06DD and PA06ED are short subroutines for polynomial evaluation and complex division, respectively. The time taken to find error bounds is usually about 25% of that taken to find the roots themselves but can rise to as much as 100% when there are many multiple roots. Because of this overhead we felt it was desirable for the user to be allowed to call PA06BD directly and instructions about how to do this are included in the specification sheet. When calling PA06CD (instruction 9 of PA06AD) the work-space provided by the user is divided into the four areas required by PA06CD. Also PA06AD checks for zero leading coefficients and sets dummy error bounds to correspond, since PA06CD assumes that the leading coefficient is non-zero.

The algorithm used by PA06BD to find the roots has already been explained in section 2 with the minor exceptions of the inclusion of tests for zero leading coefficients (roots at infinity), tests for zero trailing coefficients (roots at zero) and the scaling of all deflated polynomials so that the largest coefficient has modulus approximately equal to the reciprocal of the modulus of the smallest non-zero coefficient. We implicitly assumed in section 2 that leading and trailing coefficients were non-zero and the inclusion of scaling minimizes the likelihood of underflow or overflow. To avoid additional roundoff we scale by a power of the floating-point base. For speed of execution (in function calls rather than basic arithmetic) we use single-length working for scaling and for finding the initial iterate. Also we avoid time-wasting evaluations of the moduli of complex numbers by working with the squares of the moduli or the sums of the absolute values of the real and imaginary parts. Subroutines PA06DD and PA06ED, called from several places in PA06BD, should be regarded as part of PA06BD. PA06DD evaluates a complex polynomial at a complex point and finds the square of the modulus of the result. PA06ED is a simple subroutine for complex division. The arguments of PA06DD and PA06ED are explained in comments at the heads of each subroutine. Other coding details of PA06BD are explained in comments.

Subroutine PA06CD, which finds the error bounds using the algorithm of section 3, again makes use of single-length arithmetic whenever possible. In particular we found that CABS executes significantly faster than its double-precision equivalent (which in any case is not standard FORTRAN). Therefore the error polynomial is held in single-length, the function k(r) given by equation (3.9) is calculated in single-length and all the Rouché tests are performed in single-length. The details of the code are explained in comments.

We have tried to make the code for the case with real coefficients, namely subroutines PA07AD-PA07ED, resemble that for the complex case as much as possible, using the same labels and the same comments wherever this is appropriate. The only significant change in PA07AD lies in the way the work-array W is subdivided when PA07CD is called. The only significant changes in PA07BD lie in the code for deflation (instructions 138-166), which is much more complicated because we need to test for a real root and need to include code for deflation with a pair of complex conjugate roots. The code for polynomial evaluation (in PA07DD) is longer because we make use of the fact that the coefficients are real and treat separately the case where the point of evaluation is real. The main changes in PA07CD are

(i) Recognising complex roots when forming the polynomial $\Pi(z-\alpha_i)$ and multiplying them in as a complex conjugate pair to preserve real coefficients (instructions 18-28). It is assumed that conjugate pairs are adjacent in array R.

(ii) Much more complicated code (instructions 39-59) for finding distances from the I^{th} root to all the rest. This code avoids calling CABS to find the distances between two real roots and so is much faster where most roots are real.

(iii) Using the same error bound for a complex root and its conjugate (instructions 114-115).

6. Test results and comparisons with other methods

Our original reason for providing a new routine for the Harwell library was that the existing routine PA01 sometimes gave incorrect answers. The new routine (PA07) is able to get answers all of which have good accuracy in about half the time. We did not pursue the error in PA01.

We have compared our algorithm with that of Jenkins and Traub (1970) in two ways. First we compared the Algol-W code given by Madsen (1973) with an Algol-W version of the code of Jenkins (1969) and found Madsen's to be 4 to 5 times faster. The required changes from Algol-60 to Algol-W are very minor. Next we compared our Fortran code with a rather free translation into Fortran of Jenkins' code. We did not try to make a literal translation but rather to use his ideas to produce efficient Fortran code. The resulting program executed between 2 and 4 times slower than ours. A further advantage of our algorithm is that it is simpler and so the code is less bulky (the object code being about $^2/_3$ as long). The accuracy of the roots produced by the two algorithms was very comparable except in the last test shown in Table 1 (Jenkins' 6th example) where we obtained much smaller errors (all less than 2×10^{-14}) because the roots of modulus 0.9 were not all found before those of modulus 1 and therefore ill-conditioned deflated polynomials were not generated. To compare the error bounds with those of Jenkins we ran the examples documented in his report. In his 6th example our bounds were much better simply because the roots had been found so much more accurately. In his 5th example (a complex polynomial of degree 21 having roots of multiplicities 1,2 and 3) several of his bounds were quite unrealistic and our bounds were all better, most of them by factors over 100 and three (or seven if multiplicities are included) by factors over 1,000. In the remaining examples the differences between the bounds were not severe.

Special purpose subroutines exist in the Harwell library for solving real cubic and quadratic equations (PA03A/AD and PA05A/AD, respectively). They use direct methods involving only the extraction of square and cube roots but sometimes they lose accuracy through unnecessary cancellation. We had hoped that a direct call of PA07BD might be nearly as fast so that

we could withdraw the special purpose subroutines, but the speed difference is by a factor of about 7. It is hoped that better versions of PA03A/AD and PA05A/AD will be written for the library in due course. Extra tests for ensuring reliability are likely to slow down the programs a little, but it seems likely that good programs significantly faster than PA07BD can be written for these special cases.

To test our subroutines we read in sets of roots and a relative error level. We used extended precision arithmetic to construct the corresponding polynomial and then made pseudo-random perturbations to its coefficients at the required relative level. The polynomial and its errors were then handed to our subroutines. This enables us to check the actual errors against the computed error discs. We began by using all the polynomials of Jenkins (1969) and about as many others from local sources, but eventually reduced our test set to those shown in Table 1 (and a few trivial ones designed to explore corners in our code) because the remainder showed no useful additional information. We used the same test data for the real case by adding an additional root consisting of the complex conjugate of any complex root.

Our test results are summarised in Table 1. For each example we show the errors and error bounds obtained for the first and last root found and one intermediate one. This gives the reader an indication of our success in finding the roots in order and we have been able to choose the intermediate root displayed so that the three roots together indicate the full range of conditioning. We also show the times (370/168 secs) for a call of PA06BD/7BD (roots only) and PA06AD/7AD (roots and bounds). It can be seen that finding the bounds usually involves quite a small overhead. The last two cases are exceptional because of the large number of roots which required discs containing many other roots. Such cases are relatively slow

because we try to find a disc with only one root, then try with two, and so on.

It was the complex version of the last example which led us to abandon finding disjoint discs, because some of the roots (e.g. (0,0.9)) are so ill-conditioned that the best disc obtainable covers all the roots. Therefore if we insist on distinct discs all that can be obtained is one disc for all the roots. By using overlapping discs, however, we obtained 12 good error bounds of which two are displayed in Table 1.

The real versions of the third and the last examples led us to abandon Peters' and Wilkinsons' running error analysis on the computed polynomial $P(z) = \Pi(z-\alpha_r)$. In the last example all the roots are well conditioned but the bounds for the errors in computing P(z) were so pessimistic that each disc contained all the roots. Example 3 was less dramatic but all the error bounds came to about 10^{-8} instead of about 10^{-14}.

TABLE 1

Test	Roots	Pert. level	Complex version					Real version				
			Order	Time	Root	Bound	Error	Order	Time	Root	Bound	Error
1	(1,1D-6),2x(-1,1D-6) (2,1D-8),2x(-2,1D-8) 5,2x3,-10	0	10	.024 .030	(-1,1D-6) 3 -10	5.4E-8 2.6E-7 4.9E-15	3.0E-10 3.8E-10 1.1E-15	16	.028 .038	(-1,0) 3 -10	7.6E-4 4.7E-7 9.2E-15	1.2E-6 4.6E-9 2.2E-15
2	±1,±1.5,±2,±2.5,...,±4 ±4.1,±4.2,±4.3,...,±4.7	1D-15	28	.11 .14	1.5 -4.2 4.7	1.1E-11 1.2E-3 3.1E-5	1.8E-13 2.2E-4 4.8E-6	28	.12 .13	1.5 4.3 4.7	1.3E-11 1.3E-3 2.4E-5	5.4E-14 1.6E-4 2.8E-6
3	Jenkins 1*: randomly chosen coefficients*	1D-13	18	.067 .087	(.158,1) (-.209,-.900) (-.885,.302)	7.0E-8 6.7E-14 1.7E-9	1.4E-8 3.1E-15 1.1E-10	36	.08 .10	(-.587,.482) (.579,-.72) (-.377,-.898)	8.8E-14 7.7E-14 3.1E-14	4.8E-15 4.6E-15 6.0E-15
4	2^i, i= -6,-5,-4,...,13	0	20	.050 .063	2^{-6} 2^2 2^{12}	2.1E-16 4.2E-13 1.0E-10	1.7E-18 4.8E-14 1.3E-11	20	.037 .038	2^{-6} 2^2 2^{13}	2.3E-16 4.9E-13 6.1E-11	1.7E-18 4.0E-14 8.2E-12
5	1D-7, 4x1, 3x2, 2x3, 4 (an expansion of Jenkins 4)	1D-10	11	.028 .044	1D-7 2 4	3.1E-17 7.5E-2 1.3E-5	8.8E-18 1.8E-2 2.6E-6	11	.014 .023	1D-7 2 4	2.2E-17 8.3E-2 9.0E-6	2.3E-18 2.7E-2 3.2E-6
6	1D6,8,9,10,...,15	0	9	.017 .019	11 14 1D6	3.3E-7 1.9E-7 5.7E-10	1.3E-9 7.7E-10 4.4E-11	9	.012 .014	11 14 1D6	3.2E-7 1.8E-7 5.7E-10	5.5E-11 4.3E-10 4.4E-11
7	5x0.1, 10x1, 2,3,4,5,6	1D-15	20	.08 .14	0.1 1 3	9.0E-4 2.9E-1 7.9E-9	2.9E-4 8.4E-2 3.1E-10	20	.04 .07	0.1 1 5	6.3E-4 2.8E-1 1.1E-9	3.9E-5 8.3E-2 4.3E-11
8	Jenkins 6*: equally spaced on semi-circles centre origin radii .9/1 (left/right half planes)	1D-14	31	.37 .72	(0,.9) (-.7794,.4500) (1,0)	2.0E-0 1.8E-3 8.4E-8	2.7E-2 1.7E-4 1.5E-8	60	.20 .28	(-.8803,-1.871) (.3090, -.9511) (-.4500,.7794)	1.7E-13 3.1E-14 1.6E-13	1.1E-15 6.0E-15 5.5E-15

*In these cases roots were input to 3/4 significant decimals.

THIS

7. Appendix. Specification sheets and listings

Harwell Subroutine Library PA06AD/BD

1. Purpose

To find all the roots of a complex polynomial

$$a_1 + a_2x+\ldots+a_{n+1}x^n$$

and error bounds for these roots.

2. Argument List

```
CALL PA06AD(A,N,R,E,W,S,NP1,LW)     (double-length standard)
CALL PA06AD(A,N,R,E,W,NP1,W)        (single-length standard)
CALL PA06AD(A,N,R,E,W)              (IBM)
```

A is a DOUBLE PRECISION (REAL for the single-length version) array of dimensions (2,n+1) which must be set by the user so that the real and imaginary parts of a_i are held in A(1,i), A(2,i). It is unaltered by the subroutine. For the IBM version A may be a COMPLEX*16 array of length n+1.

N is an INTEGER variable containing the degree n of the polynomial. Its value must be positive.

R is a DOUBLE PRECISION (REAL for the single-length version) array of dimensions (2,n) used to return the roots. These are held with real and imaginary parts in R(1,i), R(2,i), i=1,2,...,n. The dummy value (1D70,1D70) is returned for each infinite root (corresponding to a zero leading coefficient). For the IBM version R may be a COMPLEX*16 array of length n.

E. is a REAL array of dimension at least (n+1) which must be set by the user to error bounds on the coefficients, or to zero if these are accurate to machine precision. On exit the first n locations contain approximate bounds on the moduli of the errors in the roots.

W is a DOUBLE PRECISION (REAL for the single-length version) work array of dimensions (2,LW).

S (double-length standard version only) is a REAL work array of dimensions (4,LW), which may be equivalenced with W.

NP1 (standard versions only) is an INTEGER which must be set to n+1.

LW (standard versions only) is an INTEGER which must be set to at least 5n/4+2 (or 3n/2+2 for the single-length version).

3. Alternative Entry

The error analysis part of a call to PA06AD takes typically about 20% of its time. If speed is important and error bounds are not wanted then a call of the form

```
           CALL PA06BD(A,N,R,W,NP1)     (standard versions)
    or     CALL PA05BD(A,N,R,W)         (IBM version)
```

should be made. The arguments are the same as those of the main call, but W need have length only (2,n+1).

4. Method

The roots are found by the algorithm of Madsen (BIT(1973) 13, 71-75), the principal features of which are Newton iteration followed by deflation. The error bounds are found by the application of Rouché's theorem as recommended by Wilkinson (J.Inst.Maths Applics.(1971) 8, 16-35) except that discs are always taken with centres on the approximate roots and errors in multiplying out the polynomial $\Pi(x-R(I))$ are ignored. The disc for each root is such that it contains exactly the same number of approximate roots R(I) as exact roots of the true polynomial. Note that in the case of true multiple roots the corresponding approximate roots may be quite well separated but each will lie in the disc of all the others and their mean will be a good estimate of the true multiple root.

PA07AD/BD

1. Purpose

To find all the roots of a real polynomial

$$a_1+a_2x+\ldots+a_{n+1}x^n$$

and error bounds for these roots.

2. Argument List

CALL PA07AD(A,N,R,E,W,S,NP1,LW) (double-length standard)
CALL PA07AD(A,N,R,E,W,NP1,LW) (single-length standard)
CALL PA07AD(A,N,R,E,W) (IBM)

A is a DOUBLE PRECISION (REAL for the single-length version) array of length at least (n+1) which must be set by the user to contain the coefficients and is unaltered by the subroutine.

N is an INTEGER variable containing the degree n of the polynomial. Its value must be positive.

R is a DOUBLE PRECISION (REAL for the single-length version) array of dimensions (2,n) used to return the roots. These are held with real and imaginary parts in R(1,i),R(2,i),i=1,2,...,n. The dummy value (1D70,0D0) is returned for each infinite root (corresponding to a zero leading coefficient). For the IBM version R may be a COMPLEX*16 array of length n.

E is a REAL array of dimension at least (n+1) which must be set by the user to error bounds on the coefficients, or to zero if these are accurate to machine precision. On exit the first n locations contain approximate bounds on the moduli of the errors in the roots.

W is a DOUBLE PRECISION (REAL for the single-length version) work array of length LW.

S (double-length standard version only) is a REAL work array of dimensions (2,LW), which may be equivalenced with W.

NP1 (standard versions only) is an INTEGER which must be set to n+1.

LW (standard versions only) is an INTEGER which must be set to 3n/2+2 (or 3n+2 for the single-length version).

3. Alternative Entry

The error analysis part of a call to PA07AD takes typically about 20% of its time. If speed is important and error bounds are not wanted then a call of the form

```
CALL PA07BD(A,N,R,W,NP1)    (standard versions)
CALL PA07BD(A,N,R,W)        (IBM version)
```

should be made. The arguments are the same as those of the main call, but W need have length only n+1.

4. Method

The roots are found by the algorithm of Madsen (BIT(1973) 13,71-75), the principal features of which are Newton iteration followed by deflation. The error bounds are found by the application of Rouché's theorem as recommended by Wilkinson (J.Inst.Maths Applics.(1971) 8, 16-35) except that discs are always taken with centres on the approximate roots and errors in multiplying out the polynomial $\Pi(x-R(I))$ are ignored. The disc for each root is such that it contains exactly the same number of approximate roots R(I) as exact roots of the true polynomial. Note that in the case of true multiple roots the corresponding approximate roots may be quite well separated but each will lie in the disc of all the others and their mean will be a good estimate of the true multiple root.

```
        C      SUBROUTINE PAO6AD(A,N,R,E,W,  NP1,LW)                              S/
        C      SUBROUTINE PAO6AD(A,N,R,E,W)                                       I/
 1             SUBROUTINE PAO6AD(A,N,R,E,W,S,NP1,LW)
        C      REAL A(2,NP1),W(2,LW),R(2,N)                                       S/
        C      DOUPLE PRECISION A(2,1),W(2,1),R(2,N)                              I/
 2             DOUBLE PRECISION A(2,NP1),W(2,LW),R(2,N)
        C      REAL E(NP1)                                                        S/
        C      REAL E(1)                                                          I/
 3             REAL E(NP1),S(4,LW)
        C      CALL PAO6BD(A,N,R,W)                                               I/
 4             CALL PAO6BD(A,N,R,W,N+1)
        C       CHECK FOR ZERO LEADING COEFFICIENTS
 5             M=N+1
 6      5      M=M-1
        C      IF( ABS(A(1,M+1))+ ABS(A(2,M+1)).EQ.0.0 .AND. M.GT.0)GO TO 5      S/
 7             IF(DABS(A(1,M+1))+DABS(A(2,M+1)).EQ.0.0 .AND. M.GT.0)GO TO 5
        C      N2=N/2                                                             S/
 8             N4=N/4
        C      IF(M.GE.1)CALL PAO6CD(A,M,M+1,R,E,W,W(1,N+2),W,W(1,N2+2))         S/
        C      IF(M.GE.1)CALL PAO6CD(A,M,M+1,R,E,W,W(1,N+2),W,W(1,N4+2))         I/
 9             IF(M.GE.1)CALL PAO6CD(A,M,M+1,R,E,W,S(1,N+2),S,S(1,N4+2))
        C  SET DUMMY DISCS CORRESPONDING TO INFINITE ROOTS
10             IF(M.EQ.N)GO TO 20
11             M=M+1
12             DO 10 I=M,N
        C10    E(I)= ABS(R(1,I))+ ABS(R(2,I))                                     S/
13       10    E(I)=DABS(R(1,I))+DABS(R(2,I))
14      20     CONTINUE
15             RETURN
16             END
```

PROGRAM UNIT PA06AD

ARGUMENTS A N R E W S NP1 LW

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	A A2	1	2	4	7	9			
DABS	ED	IF		7	13						
E	ER	V	ASA1	1	3	9	13				
I	I	V	SS	12	13						
LW	I	V	A S	1	2	3					
M	I	V	SS	5	6	7	9	10	11	12	
N	I	V	A S	1	2	4	5	8	9	10	12
NP1	I	V	A S	1	2	3					
N4	I	V	SS	8	9						
PA06BD		SN		4							
PA06CD		SN		9							
R	ED	V	A A2	1	2	4	9	13			
S	ER	V	A A2	1	3	9					
W	ED	V	A A2	1	2	4	9				
10				12	13						
20				10	14						
5				6	7						

```
C      SUBROUTINE PA06BD(A1,M,ROOT,A)                                    I/
       SUBROUTINE PA06BD(A1,M,ROOT,A,MP1)
C      REAL ZO(2),FOZ(2),Z(2),DZ(2),F1Z(2),FZ(2),W(2),                  S/
C     1FW(2),A(2,MP1),A1(2,MP1),ROOT(2,M),DZK(2)                        S/
C      DOUBLE PRECISION ZO(2),FOZ(2),Z(2),DZ(2),F1Z(2),FZ(2),W(2),      I/
C     1FW(2),A(2,1),A1(2,1),ROOT(2,M),DZK(2)                            I/
       DOUBLE PRECISION ZO(2),FOZ(2),Z(2),DZ(2),F1Z(2),FZ(2),W(2),
      1FW(2),A(2,MP1),A1(2,MP1),ROOT(2,M),DZK(2)
C ALL THE ABOVE ARRAYS HOLD COMPLEX VARIABLES AND COMPLEX ARRAYS AND
C WILL BE REFERRED TO IN COMMENTS AS SUCH, WITH NO MENTION OF THE FIRST
C SUBSCRIPT
C
C        WHEN (M-N) ZEROS ARE FOUND,  A(1), ... ,A(N+1),  WILL HOLD THE
C        COEFFICIENTS OF THE DEFLATED POLYNOMIAL.  ROOT(N+1),...,ROOT(M)
C        WILL HOLD THE ZEROS FOUND.    A1(1), ... ,A1(N)   WILL HOLD THE
C        COEFFICIENTS OF THE DERIVATIVE OF THE DEFLATED POLYNOMIAL.
C        ROOT(1),....,ROOT(N),A1(N+1),....,A1(M+1) HOLD COEFFICIENTS OF
C         THE ORIGINAL POLYNOMIAL.
C      REAL FO,FF,F,FA,PA06DD,DZ1                                       S/
       DOUBLE PRECISION FO,FF,F,FA,PA06DD,DZ1
       LOGICAL STAGE1,DIV2
C STAGE1 IS .TRUE. DURING STAGE1 OF THE ITERATION AND .FALSE.
C DURING STAGE2. DIV2 IS .TRUE. IF THE SEARCH WITH STEP-LENTHS
C DZ,DZ/2,... IS IN USE AND .FALSE. IF THE SEARCH WITH STEPS DZ,2*DZ,...
C IS IN USE.
C      DATA BIG/1.E70/,SMALL/1.E-70/,BASE/16./,EPS/1.E-6/               S/
       DATA BIG/1.E70/,SMALL/1.E-70/,BASE/16./,EPS/2.3E-16/
C BIG,SMALL ARE NUMBERS NEAR OVERFLOW/UNDERFLOW LIMITS. BASE IS THE BASE
C OF THE FLOATING POINT ARITHMETIC IN USE. EPS IS THE SMALLEST
C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
C IN USE.
       SSMALL=SQRT(SMALL)
       ALOGB=ALOG(BASE)
       N=M
C
C STORE ORIGINAL POLYNOMIAL IN A AND IN ROOT.
       J=M+1
       A(1,J)=A1(1,1)
       A(2,J)=A1(2,1)
       DO 10 I=1,M
       ROOT(1,I)=A1(1,I)
       ROOT(2,I)=A1(2,I)
       A(1,I)=A1(1,J)
```

```
      A(2,I)=A1(2,J)
   10 J=J-1
C
C       TEST FOR ZEROS AT INFINITY
C20   IF( ABS(A(1,1))+ ABS(A(2,1)).GT.0.0)GO TO 40                    S/
   20 IF(DABS(A(1,1))+DABS(A(2,1)).GT.0.0)GO TO 40
      DO 30 I=1,N
      A(1,I)=A(1,I+1)
   30 A(2,I)=A(2,I+1)
      ROOT(1,N)=BIG
      ROOT(2,N)=BIG
      N=N-1
      IF (N.GT.0) GOTO 20
      GOTO 310
   40 IF (N.LE.1) GOTO 260
      N1=N+1
C
C SCALE THE COEFFICIENTS
      U1=0.
      U2=BIG
      DO 50 K=1,N1
C     U= ABS(A(1,K))+ ABS(A(2,K))                                     S/
      U=DABS(A(1,K))+DABS(A(2,K))
      IF(U.LE.0.)GO TO 50
      IF (U.GT.U1) U1=U
      IF (U.LT.U2) U2=U
   50 CONTINUE
      U=SQRT(U1)*SQRT(U2)
      I=-ALOG(U)/ALOGB
      U=BASE**I
      DO 70 K=1,N
      A(1,K)=A(1,K)*U
      A(2,K)=A(2,K)*U
      A1(1,K)=A(1,K)*FLOAT(N1-K)
   70 A1(2,K)=A(2,K)*FLOAT(N1-K)
      A(1,N1)=A(1,N1)*U
      A(2,N1)=A(2,N1)*U
C
C       TEST FOR ZEROS AT (0.,0.)
      Z(1)=0.
      Z(2)=0.
C     IF( ABS(A(1,N1))+ ABS(A(2,N1)).LE.SSMALL)GO TO 290              S/
      IF(DABS(A(1,N1))+DABS(A(2,N1)).LE.SSMALL)GO TO 290
      Z0(1)=0.
```

```
      ZO(2)=0.
      FO=A(1,N1)**2+A(2,N1)**2
      FMIN=FO*(FLOAT(N)*16.*EPS)**2
C
C THE FOLLOWING QUANTITIES ARE HELD AT THE START OF EACH ITERATION
C Z IS THE CURRENT POINT,  F=CABS(F(Z))**2
C ZO IS THE LAST POINT,  FOZ=F'(ZO),  FO=CABS(F(ZO))**2
C RO=3*CABS(Z-ZO)
C DZ IS THE LAST TENTATIVE STEP IF THE LAST ITERATION WAS SUCCESSFUL OR
C    ELSE IS THE REQUIRED NEXT TENTATIVE STEP. ON FIRST ITERATION IT IS
C    SET TO Z.
C FF=CABS(F(ZT))**2 WHERE ZT IS THE LAST TENTATIVE POINT.
C
C SET INITIAL ITERATES AND QUANTITY USED IN THE CONVERGENCE TEST.
      FF=FO
      U0=FO
      T=BIG
      DO 80 K=1,N
      U=A(1,K)**2+A(2,K)**2
      IF (U.EQ.0.) GOTO 80
      U=ALOG(U0/U)/FLOAT(2*(N1-K))
      IF (U.LT.T) T=U
   80 CONTINUE
      T=EXP(T)
      FOZ(1)=A(1,N)
      FOZ(2)=A(2,N)
      Z(1)=1.
      Z(2)=0.
C     IF( ABS(FOZ(1))+ ABS(FOZ(2)).LE.0.)GO TO 100                   S/
      IF(DABS(FOZ(1))+DABS(FOZ(2)).LE.0.)GO TO 100
C     Z=-A(N1)/A(N)
      CALL PA06ED(A(1,N1),A(1,N),Z)
C100  U=0.5*T/( ABS(Z(1))+ ABS(Z(2)))                                S/
 100  U=0.5*T/(DABS(Z(1))+DABS(Z(2)))
      Z(1)=Z(1)*U
      Z(2)=Z(2)*U
      DZ(1)=Z(1)
      DZ(2)=Z(2)
      F=PA06DD(Z,FZ,N+1,A)
      RO=0.5*T
C
C        CALCULATION OF THE TENTATIVE STEP DZ AND WHETHER IN STAGE1.
C THIS IS WHERE THE ITERATION STARTS IF THE PREVIOUS ONE WAS SUCCESSFUL.
 120  U=PA06DD(Z,F1Z,N  ,A1)
```

```
      IF (U.EQ.0.) GOTO 140
C     DZ=-FZ/F1Z
      CALL PA06ED(FZ,F1Z,DZ)
      F2=((FOZ(1)-F1Z(1))**2+(FOZ(2)-F1Z(2))**2)/
     1   ((ZO(1)-Z(1))**2+(ZO(2)-Z(2))**2)
      STAGE1=F*F2/U.GT.U*0.25 .OR. F.NE.FF
C     R= ABS(DZ(1))+ ABS(DZ(2))                                  S/
      R=DABS(DZ(1))+DABS(DZ(2))
      IF(R.LE.RO*3.)GO TO 150
      DZ1=DZ(1)
      DZ(1)=(DZ1*1.8-DZ(2)*2.4)*RO/R
      DZ(2)=(DZ1*2.4+DZ(2)*1.8)*RO/R
      GOTO 150
140   DZ1=DZ(1)
      DZ(1)= DZ1*1.8-DZ(2)*2.4
      DZ(2)= DZ1*2.4+DZ(2)*1.8
      STAGE1=.TRUE.
150   FOZ(1)=F1Z(1)
      FOZ(2)=F1Z(2)
C
C  FIND NEXT POINT IN THE ITERATION. THIS IS WHERE THE ITERATION STARTS
C IF THE PREVIOUS ONE WAS UNSUCCESSFUL.
160   ZO(1)=Z(1)
      ZO(2)=Z(2)
      FO=F
      DZK(1)=DZ(1)
      DZK(2)=DZ(2)
      Z(1)=ZO(1)+DZ(1)
      Z(2)=ZO(2)+DZ(2)
C     IF EITHER PART OF Z IS SMALL REPLACE BY ZERO TO AVOID UNDERFLOWS
C     IF( ABS(Z(1)).LT.EPS* ABS(Z(2)))Z(1)=0.                    S/
      IF(DABS(Z(1)).LT.EPS*DABS(Z(2)))Z(1)=0.
C     IF( ABS(Z(2)).LT.EPS* ABS(Z(1)))Z(2)=0.                    S/
      IF(DABS(Z(2)).LT.EPS*DABS(Z(1)))Z(2)=0.
      W(1)=Z(1)
      W(2)=Z(2)
      F=PA06DD(Z,FZ,N+1,A)
      FF=F
      IF (.NOT.STAGE1) GOTO 240
C BEGINNING OF STAGE1 SEARCH.
      J=1
      DIV2=F.GE.FO
  180 IF (DIV2) GOTO 190
      W(1)=W(1)+DZ(1)
```

```
      W(2)=W(2)+DZ(2)
      GOTO 200
  190 DZ(1)=DZ(1)*0.5
      DZ(2)=DZ(2)*0.5
      W(1)=ZO(1)+DZ(1)
      W(2)=ZO(2)+DZ(2)
  200 FA=PA06DD(W,FW,N+1,A)
      IF (FA.GE.F) GOTO 240
      F=FA
      FZ(1)=FW(1)
      FZ(2)=FW(2)
      Z(1)=W(1)
      Z(2)=W(2)
      J=J+1
      IF (DIV2.AND.(J.EQ.3)) GOTO 220
      IF (J.LE.N) GOTO 180
      GOTO 240
  220 DZ1=DZ(1)
      DZ(1)=DZ1*0.6-DZ(2)*0.8
      DZ(2)=DZ1*0.8+DZ(2)*0.6
      Z(1)=ZO(1)+DZ(1)
      Z(2)=ZO(2)+DZ(2)
      F=PA06DD(Z,FZ,N+1,A)
C END OF STAGE1 SEARCH
C240  RO= ABS(ZO(1)-Z(1))+ ABS(ZO(2)-Z(2))                                S/
  240 RO=DABS(ZO(1)-Z(1))+DABS(ZO(2)-Z(2))
C
C CONVERGENCE TEST.
      IF(F.LT.FO)GO TO 250
      Z(1)=ZO(1)
      Z(2)=ZO(2)
C250  R1= ABS(Z(1))+ ABS(Z(2))                                             S/
  250 R1=DABS(Z(1))+DABS(Z(2))
      IF(RO.LT.EPS*R1)GO TO 270
      IF (F.LT.FO) GOTO 120
      F=FO
      IF(F.LE.FMIN)GO TO 270
      DZ(1)=DZK(1)*(-0.3)-DZK(2)*(-0.4)
      DZ(2)=DZK(1)*(-0.4)+DZK(2)*(-0.3)
      STAGE1=.TRUE.
      GOTO 160
C
C DEAL WITH N=1 CASE
C     Z=-A(2)/A(1)
```

```
260   CALL PA06ED(A(1,2),A(1,1),Z)
      GO TO 290
C
C DEFLATE, STORE ROOT, RESTORE COEFFICIENT OF ORIGINAL POLY AND REDUCE N
270   DO 280 K=2,N
      A(1,K)=A(1,K-1)*Z(1)-A(2,K-1)*Z(2)+A(1,K)
280   A(2,K)=A(1,K-1)*Z(2)+A(2,K-1)*Z(1)+A(2,K)
290   A1(1,N)=ROOT(1,N)
      A1(2,N)=ROOT(2,N)
      ROOT(1,N)=Z(1)
      ROOT(2,N)=Z(2)
      N=N-1
      IF(N-1)310,260,40
310   RETURN
      END
```

PROGRAM UNIT PA068D

ARGUMENTS A1 M ROOT A MP1

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	ASA2	1	2	10	11	15	16	18	20
				21	32	41	42	43	44	45	46
				49	52	58	64	65	69	75	105
				118	134	148	151	152			
ALOG	ER	FN		7	38	60					
ALOGB	R	V	SS	7	38						
A1	ED	V	ASA2	1	2	10	11	13	14	15	16
				43	44	77	153	154			
BASE	R	V	SS	5	7	39					
BIG	R	V	SS	5	22	23	30	56			
DABS	ED	IF		18	32	49	68	70	82	101	102
				135	139						
DIV2	EL	V	SS	4	109	110	126				
DZ	ED	V	SA1	2	73	74	79	82	84	85	86
				88	89	90	97	98	99	100	111
				112	114	115	116	117	129	130	131
				132	133	144	145				
DZK	ED	V	SA1	2	97	98	144	145			
DZ1	ED	V	SS	3	84	85	86	88	89	90	129
				130	131						
EPS	R	V	SS	5	53	101	102	140			
EXP	ER	FN		63							
F	ED	V	SS	3	75	81	96	105	106	109	119
				120	134	136	141	142	143		
FA	ED	V	SS	5	118	119	120				
FF	ED	V	SS	3	54	81	106				
FLOAT	R	IF		43	44	53	60				
FMIN	R	V	SS	53	143						
FW	ED	V	A1	2	118	121	122				
FZ	ED	V	SA1	2	75	79	105	121	122	134	
FO	ED	V	SS	3	52	53	54	55	96	109	136
				141	142						
FOZ	ED	V	SA1	2	64	65	68	80	92	93	
F1Z	ED	V	A1	2	77	79	80	92	93		
F2	R	V	SS	80	81						

THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

36-37

```
      SUBROUTINE PA06CD(A,N,NP1,R,E,W,F,IG,CR)
C IG AND CR MAY BE DYNAMICALLY EQUIVALENCED WITH SEPARATE PARTS OF W
C     REAL A(2,NP1),R(2,N),W(2,NP1),C(2)                              S/
      DOUBLE PRECISION A(2,NP1),R(2,N),W(2,NP1),C(2)
      REAL E(NP1),F(NP1),CR(N),Z(2),AI(2),IG(N)
C W IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
C OF THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS.
C F IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
C OF THE ERROR POLYNOMIAL.
C IG IS A WORKSPACE ARRAY OF LENGTH N USED TO LINK TOGETHER THE ROOTS
C IN A GROUP. THE FIRST IS IGI, IG(K) FOLLOWS K AND THE LAST IS I.
C OTHER ROOTS HAVE IG(K)=0.
C CR IS A WORKSPACE ARRAY OF LENGTH N USED TO HOLD DISTANCES FROM ROOT I
C TO TH REST.
C     DATA EPS/1.E-6/,BIG/1.E70/                                      S/
      DATA EPS/2.3E-16/,BIG/1.E70/
C BIG IS A NUMBER NEAR THE OVERFLOW LIMIT. EPS IS THE SMALLEST
C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
C IN USE.
      FACT=1.05**(1./FLOAT(N))
      N1=N+1
C
C MULTIPLY OUT THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS
      W(1,1)=A(1,N1)
      W(2,1)=A(2,N1)
      DO 10 I=1,N
      W(1,I+1)=0.
      W(2,I+1)=0.
      II=N1-I
      C(1)=-R(1,II)
      C(2)=-R(2,II)
      DO 10 JJ=1,I
      J=I+1-JJ
      W(1,J+1)=C(1)*W(1,J)-C(2)*W(2,J)+W(1,J+1)
   10 W(2,J+1)=C(1)*W(2,J)+C(2)*W(1,J)+W(2,J+1)
C
C FIND COEFFICIENTS OF ERROR POLYNOMIAL
      DO 20 I=1,N1
      II=N+2-I
      AI(1)=A(1,II)
      AI(2)=A(2,II)
      Z(1)=A(1,II)-W(1,I)
      Z(2)=A(2,II)-W(2,I)
```

```
20    F(I)=CABS(CMPLX(Z(1),Z(2)))+AMAX1(E(II),CABS(CMPLX(AI(1),AI(2)))
     1*EPS)
      DO 30 I=1,N
30    IG(I)=0
C
C MAIN ERROR-BOUNDING LOOP STARTS HERE AND EXTENDS TO THE END
C OF THE SUBROUTINE  IF FINDS A BOUND FOR ROOT I.
      DO 130 I=1,N
      M=1
      IG(I)=-1
      IG1=I
      C(1)=R(1,I)
      C(2)=R(2,I)
      D=0.
C
C FIND DISTANCES TO OTHER ROOTS
      DO 32 K=1,N
      Z(1)=R(1,K)-C(1)
      Z(2)=R(2,K)-C(2)
32    CR(K)=CABS(CMPLX(Z(1),Z(2)))
C
C INITIALIZATION STATEMENTS WHICH ARE ALSO NEEDED WHEN M IS INCREASED
40    DMIN=BIG
      Z(1)=C(1)
      Z(2)=C(2)
      S=CABS(CMPLX(Z(1),Z(2)))+D
      RAD=D
C TEST ROUCHE CONDITION WITH RADIUS RAD.
45    TOP=F(1)
      PROD=1.
      Z(1)=A(1,N1)
      Z(2)=A(2,N1)
      BTM=CABS(CMPLX(Z(1),Z(2)))
      DO 80 K=1,N
      TOP=S*TOP+F(K+1)
      DIST=CR(K)
      IF(IG(K).NE.0.)GO TO 70
      BTM=BTM*(DIST-RAD)
      IF(DMIN.LE.DIST)GO TO 80
      DMIN=DIST
      L=K
      GO TO 80
70    PROD=PROD*(RAD-DIST)
80    CONTINUE
```

```
      IF(BTM.EQ.0.0)GO TO 100
      TB=  ABS(TOP/BTM)
      IF(PROD.GE.TB)GO TO 110
C FIND A NEW TRIAL RADIUS.
      OLDS=S
      OLDR=RAD
      RL=RAD
      RAD=D+1.1*TB
      IF (M.EQ.1)GO TO 98
      RM=D+(1.1*TB)**(1./FLOAT(M))
C BISECTION LOOP. RL AND RM HOLD UPPER AND LOWER BOUNDS.
83    RAD=(RL+RM)/2.
      PROD=1.
      K=IG1
90    PROD=PROD*(RAD-CR(K))
      K=IFIX(IG(K))
      IF(K.GT.0)GO TO 90
      IF(PROD.GE.1.10*TB)GO TO 94
      IF(RAD.GE.DMIN)GO TO 100
      IF(PROD.GT.1.05*TB)GO TO 98
      RL=RAD
      GO TO 83
94    RM=RAD
      GO TO 83
93    S=S+RAD-OLDR
      IF(RAD.GE.DMIN)GO TO 100
      IF( S*((DMIN-OLDR)/(DMIN-RAD)).LE.FACT*OLDS)GO TO 110
      GO TO 45
C
C  ADD ROOT TO GROUP
100   M=M+1
      IG(L)=FLOAT(IG1)
      IG1=L
      D=DMIN
      GO TO 40
C
C STORE ERROR BOUND AND RESET IG
110   E(I)=RAD
120   K=IG1
      IG1=IFIX(IG(IG1))
      IG(K)=0.
      IF(K.NE.I)GO TO 120
130   CONTINUE
      RETURN
      END
```

PROGRAM UNIT PA06CD

ARGUMENTS A N NP1 R E W F IG CR

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	A A2	1	2	7	8	21	22	23	24
				46	47						
ABS	R	IF		61							
AI	ER	V	SA1	3	21	22	25				
AMAX1	R	IF		25							
BIG	R	V	SS	4	39						
BTM	R	V	SS	48	53	60	61				
C	ED	V	SA1	2	13	14	17	18	32	33	36
				37	40	41					
CABS	ER	FN		25	38	42	48				
CMPLX	EC	IF		25	38	42	46				
CR	ER	V	ASA1	1	3	38	51	72			
D	R	V	SS	34	42	43	66	68	89		
DIST	R	V	SS	51	53	54	55	58			
DMIN	R	V	SS	39	54	55	76	83	84	89	
E	ER	V	ASA1	1	3	25	91				
EPS	R	V	SS	4	25						
F	ER	V	ASA1	1	3	25	44	50			
FACT	R	V	SS	5	84						
FLOAT	R	IF		5	68	87					
I	I	V	SS	9	10	11	12	15	16	19	20
				23	24	25	26	27	28	30	31
				32	33	91	95				
IFIX	I	IF		73	93						
IG	ER	V	ASA1	1	3	27	30	52	73	87	93
				94							
IG1	I	V	SS	31	71	87	88	92	93		
II	I	V	SS	12	13	14	20	21	22	23	24
				25							
J	I	V	SS	16	17	18					
JJ	I	V	SS	15	16						
K	I	V	SS	35	36	37	38	49	50	51	52
				56	71	72	73	74	92	94	95
L	I	V	SS	56	87	88					
M	I	V	SS	29	67	68	86				

N	I	V	A S	1	2	3	5	6	9	20	26
				28	35	49					
NP1	I	V	A S	1	2	3					
N1	I	V	SS	6	7	8	12	19	46	47	
OLDR	R	V	SS	64	82	84					
OLDS	R	V	SS	63	84						
PROD	R	V	SS	45	58	62	70	72	75	77	
R	ED	V	A A2	1	2	13	14	32	33	36	37
RAD	R	V	SS	43	53	58	64	65	66	69	72
				76	78	80	82	83	84	91	
RL	R	V	SS	65	69	78					
RM	R	V	SS	68	69	80					
S	R	V	SS	42	50	63	82	84			
TB	R	V	SS	61	62	66	68	75	77		
TOP	R	V	SS	44	50	61					
W	ED	V	ASA2	1	2	7	8	10	11	17	18
				23	24						
Z	ER	V	SA1	3	23	24	25	36	37	38	40
				41	42	46	47	48			
10				9	15	18					
100				60	76	83	86				
110				62	84	91					
120				92	95						
130				28	96						
20				19	25						
30				26	27						
32				35	38						
40				39	90						
45				44	85						
70				52	58						
80				49	54	57	59				
83				69	79	81					
90				72	74						
94				75	80						
98				67	77	82					

```
             C      REAL FUNCTION PA06DD(Z,FZ,N1,A)                                          S/
       1            DOUBLE PRECISION FUNCTION PA06DD(Z,FZ,N1,A)
             C      REAL Z(2),FZ(2),A(2,N1),P,Q,R,X,Y                                        S/
       2            DOUBLE PRECISION Z(2),FZ(2),A(2,N1),P,Q,R,X,Y
             C Z AND FZ CONTAIN COMPLEX NUMBERS.
             C A CONTAINS COMPLEX POLYNOMIAL COEFFICIENTS.
             C POLYNOMIAL VALUE AT Z IS SET IN FZ AND THE SQUARE OF ITS MODULUS
             C IS RETURNED AS FUNCTION VALUE.
       3            N=N1-1
       4            X=Z(1)
       5            Y=Z(2)
       6            P=A(1,1)
       7            Q=A(2,1)
       8            DO 10 I=1,N
       9            R=P*X-Q*Y+A(1,I+1)
      10            Q=Q*X+P*Y+A(2,I+1)
      11     10     P=R
      12            FZ(1)=P
      13            FZ(2)=Q
      14            PA06DD=P*P+Q*Q
      15            RETURN
      16            END

NAME        TYPE  USE  ATTRIBUTES  REFERENCES

A            ED    V      A A2       1    2    6    7    9   10
FZ           ED    V      ASAI       1    2   12   13
I             I    V       SS        8    9   10
N             I    V       SS        3    8
N1            I    V      A S        1    2    3
P            ED    V       SS        2    6    9   10   11   12   14
PA06DD       ED    FN      S         1   14
Q            ED    V       SS        2    7    9   10   13   14
R            ED    V       SS        2    9   11
X            ED    V       SS        2    4    9   10
Y            ED    V       SS        2    5    9   10
Z            ED    V      A A1       1    2    4    5
10                                   8   11
```

```
 1          SUBROUTINE PA06ED(A,B,Q)
      C     REAL A(2),B(2),Q(2),U,A1,C,D,E                              S/
 2          DOUBLE PRECISION A(2),B(2),Q(2),U,A1,C,D,E
      C IMPLEMENTS THE COMPLEX DIVISION Q=-A/B
      C     U= ABS(B(1))+ ABS(B(2))                                     S/
 3          U=DABS(B(1))+DABS(B(2))
 4          A1=A(1)
 5          C=B(1)/U
 6          D=B(2)/U
 7          E=-(C*C+D*D)*U
 8          Q(1)=(A1*C+A(2)*D)/E
 9          Q(2)=(A(2)*C-A1*D)/E
10          RETURN
      C                                                                 //
11          END
```

```
PROGRAM UNIT     PA06ED

ARGUMENTS           A      B      Q

NAME      TYPE  USE  ATTRIBUTES REFERENCES

A          ED    V     A A1      1    2    4    8    9
A1         ED    V      SS       2    4    8    9
B          ED    V     A A1      1    2    3    5    6
C          ED    V      SS       2    5    7    8    9
D          ED    V      SS       2    6    7    8    9
DABS       ED    IF              3
E          ED    V      SS       2    7    8    9
Q          ED    V     ASA1      1    2    8    9
U          ED    V      SS       2    3    5    6    7
```

```
C     SUBROUTINE PA07AD(A,N,R,E,W,NP1,LW)                                 S/
C     SUBROUTINE PA07AD(A,N,R,E,W)                                        I/
      SUBROUTINE PA07AD(A,N,R,E,W,S,NP1,LW)
C     REAL A(NP1),W(LW),R(2,N)                                            S/
C     DOUBLE PRECISION A(1),W(1),R(2,N)                                   I/
      DOUBLE PRECISION A(NP1),W(LW),R(2,N)
C     REAL E(NP1)                                                         S/
C     REAL E(1)                                                           I/
      REAL E(NP1),S(2,LW)
C     CALL PA07BD(A,N,R,W)                                                I/
      CALL PA07BD(A,N,R,W,N+1)
C      CHECK FOR ZERO LEADING COEFFICIENTS
      M=N+1
5     M=M-1
C     IF(  ABS(A(M+1)).EQ.0.0 .AND. M.GT.0)GO TO 5                        S/
      IF( DABS(A(M+1)).EQ.0.0 .AND. M.GT.0)GO TO 5
C     N2=N*2+1                                                            S/
      N2=N/2
C     IF(M.GE.1)CALL PA07CD(A,M,M+1,R,E,W,W(N+2),W,W(N2+2))               S/
C     IF(M.GE.1)CALL PA07CD(A,M,M+1,R,E,W,W(N+2),W,W(N2+2))               I/
      IF(M.GE.1)CALL PA07CD(A,M,M+1,R,E,W,S(1,N+2),S,S(1,N2+2))
C  SET DUMMY DISCS CORRESPONDING TO INFINITE ROOTS
      IF(M.EQ.N)GO TO 20
      M=M+1
      DO 10 I=M,N
C10   E(I)= ABS(R(1,I))+ ABS(R(2,I))                                      S/
 10   E(I)=DABS(R(1,I))+DABS(R(2,I))
 20   CONTINUE
      RETURN
      END
```

PROGRAM UNIT PA07AD

ARGUMENTS A N R E W S NP1 LW

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	A A1	1	2	4	7	9			
DABS	ED	IF		7	13						
E	ER	V	ASA1	1	3	9	13				
I	I	V	SS	12	13						
LW	I	V	A S	1	2	3					
M	I	V	SS	5	6	7	9	10	11	12	
N	I	V	A S	1	2	4	5	8	9	10	12
NP1	I	V	A S	1	2	3					
N2	I	V	SS	8	9						
PA07BD		SN		4							
PA07CD		SN		9							
R	ED	V	A A2	1	2	4	9	13			
S	ER	V	A A2	1	3	9					
W	ED	V	A A1	1	2	4	9				
10				12	13						
20				10	14						
5				6	7						

```
      C      SUBROUTINE PA07BD(A1,M,ROOT,A)                                      I/
  1          SUBROUTINE PA07BD(A1,M,ROOT,A,MP1)
      C      REAL Z0(2),F0Z(2),Z(2),DZ(2),F1Z(2),FZ(2),W(2),                      S/
      C     1 FW(2),ROOT(2,M),DZK(2)                                              S/
  2          DOUBLE PRECISION Z0(2),F0Z(2),Z(2),DZ(2),F1Z(2),FZ(2),W(2),
  2         1 FW(2),ROOT(2,M),DZK(2)
      C ALL THE ABOVE ARRAYS HOLD COMPLEX VARIABLES AND COMPLEX ARRAYS AND
      C WILL BE REFERRED TO IN COMMENTS AS SUCH, WITH NO MENTION OF THE FIRST
      C SUBSCRIPT
      C
      C        WHEN (M-N) ZEROS ARE FOUND,  A(1), ... ,A(N+1),  WILL HOLD THE
      C        COEFFICIENTS OF THE DEFLATED POLYNOMIAL.  ROOT(N+1),...,ROOT(M)
      C        WILL HOLD THE ZEROS FOUND.    A1(1), ... ,A1(N)   WILL HOLD THE
      C        WILL HOLD THE ZEROS FOUND.    A1(1), ... ,A1(N)   WILL HOLD THE
      C        COEFFICIENTS OF THE DERIVATIVE OF THE DEFLATED POLYNOMIAL.
      C        ROOT(1),...,ROOT(N),A1(N+1),...,A1(M+1) HOLD COEFFICIENTS OF
      C          THE ORIGINAL POLYNOMIAL.
      C      REAL F0,FF,F,FA,PA07DD,DZ1,A1(MP1),A(MP1),FC,FD,G,AC                S/
      C     1,PP,QQ,RR,SS,TT                                                      S/
      C      DOUBLE PRECISION F0,FF,F,FA,PA07DD,DZ1,A1(  1),A(  1),FC,FD,G      I/
      C     1,AC,PP,QQ,RR,SS,TT                                                   I/
  3          DOUBLE PRECISION F0,FF,F,FA,PA07DD,DZ1,A1(MP1),A(MP1),FC,FD,G
  3         1,AC,PP,QQ,RR,SS,TT
  4          LOGICAL STAGE1,DIV2
      C STAGE1 IS .TRUE. DURING STAGE1 OF THE ITERATION AND .FALSE.
      C DURING STAGE2. DIV2 IS .TRUE. IF THE SEARCH WITH STEP-LENTHS
      C DZ,DZ/2,... IS IN USE AND .FALSE. IF THE SEARCH WITH STEPS DZ,2*DZ,...
      C IS IN USE.
      C      DATA BIG/1.E70/,SMALL/1.E-70/,BASE/16./,EPS/1.0E-6/                  S/
  5          DATA BIG/1.0E70/,SMALL/1.0E-70/,BASE/16./,EPS/2.3E-16/
      C BIG,SMALL ARE NUMBERS NEAR OVERFLOW/UNDERFLOW LIMITS. BASE IS THE BASE
      C OF THE FLOATING POINT ARITHMETIC IN USE. EPS IS THE SMALLEST
      C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
      C IN USE.
  6          SSMALL=SQRT(SMALL)
  7          ALOGB=ALOG(BASE)
  8          N=M
      C
      C STORE ORIGINAL POLYNOMIAL IN A AND IN ROOT.
  9          J=M+1
 10          A(J)=A1(1)
 11          DO 10 I=1,M
 12          ROOT(1,I)=A1(I)
```

```
      A(I)=A1(J)
 10   J=J-1
C
C       TEST FOR ZEROS AT INFINITY
C20   IF( ABS(A(1)).GT.0.)GO TO 40                                    S/
 20   IF(DABS(A(1)).GT.0.)GO TO 40
      DO 30 I=1,N
   30 A(I)=A(I+1)
      ROOT(1,N)=BIG
      ROOT(2,N)=0.
      N=N-1
      IF (N.GT.0) GOTO 20
      GOTO 310
   40 IF (N.LE.1) GOTO 260
      N1=N+1
C
C SCALE THE COEFFICIENTS
      U1=0.
      U2=BIG
      DO 50 K=1,N1
C     U= ABS(A(K))                                                    S/
      U=DABS(A(K))
      IF(U.LE.0.)GO TO 50
      IF (U.GT.U1) U1=U
      IF (U.LT.U2) U2=U
   50 CONTINUE
      U=SQRT(U1)*SQRT(U2)
      I=-ALOG(U)/ALOGB
      U=BASE**I
      DO 70 K=1,N
      A(K)=A(K)*U
   70 A1(K)=A(K)*FLOAT(N1-K)
      A(N1)=A(N1)*U
C
C       TEST FOR ZEROS AT (0.,0.)
      Z(1)=0.
      Z(2)=0.
C     IF( ABS(A(N1)).LE.SSMALL)GO TO 290                              S/
      IF(DABS(A(N1)).LE.SSMALL)GO TO 290
      ZO(1)=0.
      ZO(2)=0.
      FO=A(N1)**2
      FMIN=FO*(FLOAT(N)*16.*EPS)**2
C
```

```
C THE FOLLOWING QUANTITIES ARE HELD AT THE START OF EACH ITERATION
C Z IS THE CURRENT POINT,   F=CABS(F(Z))**2
C ZO IS THE LAST POINT,   FOZ=F'(ZO),   FO=CABS(F(ZO))**2
C RO=3*CABS(Z-ZO)
C DZ IS THE LAST TENTATIVE STEP IF THE LAST ITERATION WAS SUCCESSFUL OR
C    ELSE IS THE REQUIRED NEXT TENTATIVE STEP. ON FIRST ITERATION IT IS
C    SET TO Z.
C FF=CABS(F(ZT))**2 WHERE ZT IS THE LAST TENTATIVE POINT.
C
C SET INITIAL ITERATES AND QUANTITY USED IN THE CONVERGENCE TEST.
      FF=FO
C     UO= ABS(A(N1))                                                S/
      UO=DABS(A(N1))
      T=BIG
      DO 80 K=1,N
C     U= ABS(A(K))                                                  S/
      U=DABS(A(K))
      IF (U.EQ.0.) GOTO 80
      U=ALOG(UO/U)/FLOAT(N1-K)
      IF (U.LT.T) T=U
   80 CONTINUE
      T=EXP(T)
      FOZ(1)=A(N)
      FOZ(2)=0.
      Z(1)=0.5*T
      IF(A(N).LT.0.)Z(1)=-Z(1)
      DZ(1)=Z(1)
      DZ(2)=0.
      F=PA07DD(Z,FZ,N+1,A)
      RO=0.5*T
C
C        CALCULATION OF THE TENTATIVE STEP DZ AND WHETHER IN STAGE1.
C THIS IS WHERE THE ITERATION STARTS IF THE PREVIOUS ONE WAS SUCCESSFUL.
  120 U=PA07DD(Z,F1Z,N  ,A1)
      IF (U.EQ.0.) GOTO 140
C     DZ=-FZ/F1Z
      CALL PA07ED(FZ,F1Z,DZ)
      F2=((FOZ(1)-F1Z(1))**2+(FOZ(2)-F1Z(2))**2)/
     1   ((ZO(1)-Z(1))**2+(ZO(2)-Z(2))**2)
      STAGE1=F*F2/U.GT.U*0.25 .OR. F.NE.FF
C     R= ABS(DZ(1))+ ABS(DZ(2))                                     S/
      R=DABS(DZ(1))+DABS(DZ(2))
      IF(R.LE.RO*3.)GO TO 150
      DZ1=DZ(1)
```

```
      DZ(1)=(DZ1*1.8-DZ(2)*2.4)*RO/R
      DZ(2)=(DZ1*2.4+DZ(2)*1.8)*RO/R
      GOTO 150
140   DZ1=DZ(1)
      DZ(1)= DZ1*1.8-DZ(2)*2.4
      DZ(2)= DZ1*2.4+DZ(2)*1.8
      STAGE1=.TRUE.
150   FOZ(1)=F1Z(1)
      FOZ(2)=F1Z(2)
C
C  FIND NEXT POINT IN THE ITERATION. THIS IS WHERE THE ITERATION STARTS
C IF THE PREVIOUS ONE WAS UNSUCCESSFUL.
160   ZO(1)=Z(1)
      ZO(2)=Z(2)
      FO=F
      DZK(1)=DZ(1)
      DZK(2)=DZ(2)
      Z(1)=ZO(1)+DZ(1)
      Z(2)=ZO(2)+DZ(2)
C    IF EITHER PART OF Z IS SMALL REPLACE BY ZERO TO AVOID UNDERFLOWS
C     IF( ABS(Z(1)).LT.EPS* ABS(Z(2)))Z(1)=0.                          S/
      IF(DABS(Z(1)).LT.EPS*DABS(Z(2)))Z(1)=0.
C     IF( ABS(Z(2)).LT.EPS* ABS(Z(1)))Z(2)=0.                          S/
      IF(DABS(Z(2)).LT.EPS*DABS(Z(1)))Z(2)=0.
      W(1)=Z(1)
      W(2)=Z(2)
      F=PA07DD(Z,FZ,N+1,A)
      FF=F
      IF (.NOT.STAGE1) GOTO 240
C BEGINNING OF STAGE1 SEARCH.
      J=1
      DIV2=F.GE.FO
  180 IF (DIV2) GOTO 190
      W(1)=W(1)+DZ(1)
      W(2)=W(2)+DZ(2)
      GOTO 200
190   DZ(1)=DZ(1)*0.5
      DZ(2)=DZ(2)*0.5
      W(1)=ZO(1)+DZ(1)
      W(2)=ZO(2)+DZ(2)
200   FA=PA07DD(W,FW,N+1,A)
      IF (FA.GE.F) GOTO 240
      F=FA
      FZ(1)=FW(1)
```

```
      FZ(2)=FW(2)
      Z(1)=W(1)
      Z(2)=W(2)
      J=J+1
      IF (DIV2.AND.(J.EQ.3)) GOTO 220
      IF (J.LE.N) GOTO 180
      GOTO 240
220   DZ1=DZ(1)
      DZ(1)=DZ1*0.6-DZ(2)*0.8
      DZ(2)=DZ1*0.8+DZ(2)*0.6
      Z(1)=Z0(1)+DZ(1)
      Z(2)=Z0(2)+DZ(2)
      F=PA07DD(Z,FZ,N+1,A)
C END OF STAGE1 SEARCH
C240   R0= ABS(Z0(1)-Z(1))+ ABS(Z0(2)-Z(2))                              S/
 240  R0=DABS(Z0(1)-Z(1))+DABS(Z0(2)-Z(2))
C
C CONVERGENCE TEST.
      IF(F.LT.F0)GO TO 250
      Z(1)=Z0(1)
      Z(2)=Z0(2)
C250   R1= ABS(Z(1))+ ABS(Z(2))                                          S/
 250  R1=DABS(Z(1))+DABS(Z(2))
      IF(R0.LT.EPS*R1)GO TO 270
      IF (F.LT.F0) GOTO 120
      F=F0
      IF(F.LE.FMIN)GO TO 270
      DZ(1)=DZK(1)*(-0.3)-DZK(2)*(-0.4)
      DZ(2)=DZK(1)*(-0.4)+DZK(2)*(-0.3)
      STAGE1=.TRUE.
      GOTO 160
C
C DEAL WITH N=1 CASE
260   Z(1)=-A(2)/A(1)
      Z(2)=0.
C
C DEFLATE, STORE ROOT, RESTORE COEFFICIENT OF ORIGINAL POLY AND REDUCE N
270   A1(N)=ROOT(1,N)
      ROOT(1,N)=Z(1)
      ROOT(2,N)=0.
      IF(Z(2).EQ.0.)GO TO 277
C     AC= ABS(Z(1))                                                      S/
      AC=DABS(Z(1))
      G=0.
```

```
      FC=A(1)
      DO 273 K=2,N1
      FD=FC*Z(1)+A(K)
C     G=AC*(G+ ABS(FC))+ ABS(FD)                                       S/
      G=AC*(G+DABS(FC))+DABS(FD)
273   FC=FD
C     IF( ABS(FC).GT.2.*EPS*G+SQRT(AMIN1(F,FO)))GO TO 283              S/
      IF(DABS(FC).GT.2.*EPS*G+SQRT(SNGL(DMIN1(F,FO))))GO TO 283
C DEFLATION WITH A REAL ROOT
277   G=0.
      DO 280 K=1,N
      G=G*Z(1)+A(K)
280   A(K)=G
      GO TO 300
C DEFLATION WITH A PAIR OF COMPLEX CONJUGATE ROOTS
283   ROOT(2,N)=Z(2)
      RR=0.
      SS=0.
      PP=-Z(1)-Z(1)
      QQ=Z(1)**2+Z(2)**2
      N=N-1
      DO 287 K=1,N
      TT=A(K)-PP*RR-QQ*SS
      A(K)=TT
      SS=RR
287   RR=TT
290   A1(N)=ROOT(1,N)
      ROOT(1,N)=Z(1)
      ROOT(2,N)=-Z(2)
300   N=N-1
      IF(N-1)310,260,40
310   RETURN
      END
```

PROGRAM UNIT PA07BD

ARGUMENTS A1 M ROOT A MP1

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	ASA1	1	3	10	13	15	17	28	37
				38	39	42	45	48	51	57	60
				63	93	106	122	136	144	146	152
				153	162	163					
AC	ED	V	SS	3	142	147					
ALOG	ER	FN		7	34	53					
ALOGR	R	V	SS	7	34						
A1	ED	V	ASA1	1	3	10	12	13	38	65	138
				166							
BASE	R	V	SS	5	7	35					
BIG	R	V	SS	5	18	26	49				
DABS	ED	IF		15	28	42	48	51	70	89	90
				123	127	142	147	149			
DIV2	EL	V	SS	4	97	98	114				
DMIN1	ED	IF		149							
DZ	ED	V	SA1	2	61	62	67	70	72	73	74
				76	77	78	85	86	87	88	99
				100	102	103	104	105	117	118	119
				120	121	132	133				
DZK	ED	V	SA1	2	85	86	132	133			
DZ1	ED	V	SS	3	72	73	74	76	77	78	117
				118	119						
EPS	R	V	SS	3	46	89	90	128	149		
EXP	ER	FN		36							
F	ED	V	SS	3	63	69	84	93	94	97	107
				108	122	124	129	130	131	149	
FA	ED	V	SS	3	106	107	108				
FC	ED	V	SS	3	144	146	147	148	149		
FD	ED	V	SS	3	146	147	148				
FF	ED	V	SS	3	47	69	94				
FLOAT	R	IF		38	46	53					
FMIN	R	V	SS	46	131						
FW	ED	V	A1	3	106	109	110				
FZ	ED	V	SA1	2	63	67	93	109	110	122	
FO	ED	V	SS	3	45	46	47	84	97	124	129

				130	149						
FOZ	ED	V	SA1	2	57	58	68	80	81		
F1Z	ED	V	A1	2	65	67	68	80	81		
F2	R	V	SS	68	69						
G	ED	V	SS	3	143	147	149	150	152	153	
I	I	V	SS	11	12	13	16	17	34	35	
J	I	V	SS	9	10	13	14	96	113	114	115
K	I	V	SS	27	28	36	37	38	50	51	53
				145	146	151	152	153	161	162	163
M	I	V	A S	1	2	8	9	11			
MP1	I	V	A S	1	3						
N	I	V	SS	8	16	18	19	20	21	23	24
				36	46	50	57	60	63	65	93
				106	115	122	138	139	140	151	155
				160	161	166	167	168	169	170	
N1	I	V	SS	24	27	38	39	42	45	48	53
				145							
PA07DD	ED	FN		3	63	65	93	106	122		
PA07ED		SN		67							
PP	ED	V	SS	3	158	162					
QQ	ED	V	SS	3	159	162					
R	R	V	SS	70	71	73	74				
ROOT	ED	V	ASA2	1	2	12	18	19	138	139	140
				155	166	167	168				
RR	ED	V	SS	3	156	162	164	165			
R0	R	V	SS	64	71	73	74	123	128		
R1	R	V	SS	127	128						
SMALL	R	V	SS	5	6						
SNGL	R	IF		149							
SQRT	ER	FN		6	33	149					
SS	ED	V	SS	3	157	162	164				
SSMALL	R	V	SS	6	42						
STAGE1	EL	V	SS	4	69	79	95	134			
T	R	V	SS	49	54	56	59	64			
TT	ED	V	SS	3	162	163	165				
U	R	V	SS	28	29	30	31	33	34	35	37
				39	51	52	53	54	65	66	69
U0	R	V	SS	48	53						
U1	R	V	SS	25	30	33					
U2	R	V	SS	26	31	33					
W	ED	V	SA1	2	91	92	99	100	104	105	106
				111	112						
Z	ED	V	SA1	2	40	41	59	60	61	63	65
				68	82	83	87	88	89	90	91

				92	93	111	112	120	121	122	123
				125	126	127	136	137	139	141	142
				146	152	155	158	159	167	168	
ZO	ED	V	SA1	2	43	44	68	82	83	87	88
				104	105	120	121	123	125	126	
10				11	14						
120				65	129						
140				66	76						
150				71	75	80					
160				82	135						
180				98	115						
190				98	102						
20				15	21						
200				101	106						
220				114	117						
240				95	107	116	123				
250				124	127						
260				23	136	170					
270				128	131	138					
275				145	148						
277				141	150						
280				151	153						
283				149	155						
287				161	165						
290				42	166						
30				16	17						
300				154	169						
310				22	170	171					
40				15	23	170					
50				27	29	32					
70				36	38						
80				50	52	55					

```
      SUBROUTINE PA07CD(A,N,NP1,R,E,W,F,IG,CR)
C IG AND CR MAY BE DYNAMICALLY EQUIVALENCED WITH SEPARATE PARTS OF W
C     REAL A(NP1),R(2,N),W(NP1),C,RS,RP                                S/
      DOUBLE PRECISION A(NP1),R(2,N),W(NP1),C,RS,RP
      REAL E(NP1),F(NP1),CR(N),IG(N)
C W IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
C OF THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS.
C F IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
C OF THE ERROR POLYNOMIAL.
C IG IS A WORKSPACE ARRAY OF LENGTH N USED TO LINK TOGETHER THE ROOTS
C IN A GROUP. THE FIRST IS IG1, IG(K) FOLLOWS K AND THE LAST IS I.
C OTHER ROOTS HAVE IG(K)=0.
C CR IS A WORKSPACE ARRAY OF LENGTH N USED TO HOLD DISTANCES FROM ROOT I
C TO TH REST.
C     DATA EPS/1.0E-6 /,BIG/1.E70/                                     S/
      DATA EPS/2.3E-16/,BIG/1.0E70/
C BIG IS A NUMBER NEAR THE OVERFLOW LIMIT. EPS IS THE SMALLEST
C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
C IN USE.
      FACT=1.05**(1./FLOAT(N))
      N1=N+1
C
C MULTIPLY OUT THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS
      DO 5 I=1,N1
    5 W(I)=0.
      W(1)=A(N1)
      I=1
    8 II=N1-I
      IF(R(2,II).NE.0.)GO TO 12
      C=R(1,II)
      DO 10 JJ=1,I
      J=I+1-JJ
   10 W(J+1)=W(J+1)-C*W(J)
      GO TO 18
   12 RS=R(1,II)+R(1,II)
      RP=R(1,II)**2+R(2,II)**2
      I=I+1
      DO 16 JJ=1,I
      J=I-JJ
      W(J+2)=W(J+2)-RS*W(J+1)
      IF(J.EQ.0)GO TO 16
      W(J+2)=W(J+2)+RP*W(J)
   16 CONTINUE
```

```
18    I=I+1
      IF(I.LE.N)GO TO 8
C
C FIND COEFFICIENTS OF ERROR POLYNOMIAL
      DO 20 I=1,N1
      II=N+2-I
C20   F(I)= ABS(A(II)-W(I))+AMAX1(E(II),ABS(A(II))*EPS)              S/
 20   F(I)=DABS(A(II)-W(I))+AMAX1(E(II),ABS(SNGL(A(II)))*EPS)
      DO 30 I=1,N
30    IG(I)=0.
C
C MAIN ERROR-BOUNDING LOOP STARTS HERE AND EXTENDS TO THE END
C OF THE SUBROUTINE  IF FINDS A BOUND FOR ROOT I.
      I=1
31    M=1
      IG(I)=-1
      IG1=I
      D=0.
C.
C FIND DISTANCES TO OTHER ROOTS
      R1=R(1,I)
      R2=R(2,I)
      IF(R2.EQ.0.)GO TO 34
      CRI=CABS(CMPLX(R1,R2))
      DO 32 K=1,N
      R1=R(1,K)-R(1,I)
      R2=R(2,K)-R(2,I)
32    CR(K)=CABS(CMPLX(R1,R2))
      GO TO 40
34    K=1
      CRI=ABS(R1)
36    R2=R(2,K)
      R1=R(1,K)-R(1,I)
      IF(R2.EQ.0.)GO TO 37
      CR(K)=CABS(CMPLX(R1,R2))
      CR(K+1)=CR(K)
      K=K+1
      GO TO 38
37    CR(K)=ABS(R1)
38    K=K+1
      IF(K.LE.N)GO TO 36
C
C INITIALIZATION STATEMENTS WHICH ARE ALSO NEEDED WHEN M IS INCREASED
40    DMIN=BIG
```

```
      S=CRI+D
      RAD=D
C TEST ROUCHE CONDITION WITH RADIUS RAD.
45    TOP=F(1)
      PROD=1.
C     BTM= ABS(A(N1))
      BTM=DABS(A(N1))
      DO 80 K=1,N
      TOP=S*TOP+F(K+1)
      DIST=CR(K)
      IF(IG(K).NE.0.)GO TO 70
      BTM=BTM*(DIST-RAD)
      IF(DMIN.LE.DIST)GO TO 80
      DMIN=DIST
      L=K
      GO TO 80
70    PROD=PROD*(RAD-DIST)
80    CONTINUE
      IF(BTM.EQ.0.0)GO TO 100
      TB=  ABS(TOP/BTM)
      IF(PROD.GE.TB)GO TO 110
C FIND A NEW TRIAL RADIUS.
      OLDS=S
      OLDR=RAD
      RL=RAD
      RAD=D+1.1*TB
      IF (M.EQ.1)GO TO 98
      RM=D+(1.1*TB)**(1./FLOAT(M))
C BISECTION LOOP. RL AND RM HOLD UPPER AND LOWER BOUNDS.
83    RAD=(RL+RM)/2.
      PROD=1.
      K=IG1
90    PROD=PROD*(RAD-CR(K))
      K=IFIX(IG(K))
      IF(K.GT.0)GO TO 90
      IF(PROD.GE.1.10*TB)GO TO 94
      IF(RAD.GE.DMIN)GO TO 100
      IF(PROD.GT.1.05*TB)GO TO 98
      RL=RAD
      GO TO 83
94    RM=RAD
      GO TO 83
98    S=S+RAD-OLDR
      IF(RAD.GE.DMIN)GO TO 100
```

S/

```
      IF( S*((DMIN-OLDR)/(DMIN-RAD)).LE.FACT*OLDS)GO TO 110
      GO TO 45
C
C  ADD ROOT TO GROUP
100   M=M+1
      IG(L)=FLOAT(IG1)
      IG1=L
      D=DMIN
      GO TO 40
C
C STORE ERROR BOUND AND RESET IG
110   E(I)=RAD
120   K=IG1
      IG1=IFIX(IG(IG1))
      IG(K)=0.
      IF(K.NE.I)GO TO 120
      IF(R(2,I).EQ.0.)GO TO 130
      E(I+1)=E(I)
      I=I+1
130   I=I+1
      IF(I.LE.N)GO TO 31
      RETURN
      END
```

PROGRAM UNIT PA07CD

ARGUMENTS A N NP1 R E W F IG CR

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	A A1	1	2	9	31	65			
ABS	R	IF		31	49	57	78				
AMAX1	R	IF		31							
BIG	R	V	SS	4	60						
BTM	R	V	SS	65	70	77	78				
C	ED	V	SS	2	13	16					
CABS	ER	FN		42	46	53					
CMPLX	EC	IF		42	46	53					
CR	ER	V	ASA1	1	3	46	53	54	57	68	89
CRI	R	V	SS	42	49	61					
D	R	V	SS	38	61	62	63	65	106		
DABS	ED	IF		31	65						
DIST	R	V	SS	68	70	71	72	75			
DMIN	R	V	SS	60	71	72	93	100	101	106	
E	ER	V	ASA1	1	3	31	108	114			
EPS	R	V	SS	4	31						
F	ER	V	ASA1	1	3	31	63	67			
FACT	R	V	SS	5	101						
FLOAT	R	IF		5	85	104					
I	I	V	SS	7	8	10	11	14	15	20	21
				22	27	28	29	30	31	32	33
				34	36	37	39	40	44	45	51
				108	112	113	114	115	116	117	
IFIX	I	IF		90	110						
IG	ER	V	ASA1	1	3	33	36	69	90	104	110
				111							
IG1	I	V	SS	37	88	104	105	109	110		
II	I	V	SS	11	12	13	18	19	30	31	
J	I	V	SS	15	16	22	23	24	25		
JJ	I	V	SS	14	15	21	22				
K	I	V	SS	43	44	45	46	48	50	51	53
				54	55	57	58	59	66	67	68
				69	73	88	89	90	91	109	111
				112							
L	I	V	SS	73	104	105					

M	I	V	SS	35	84	85	103				
N	I	V	A S	1	2	3	5	6	28	30	32
				43	59	66	117				
NP1	I	V	A S	1	2	3					
N1	I	V	SS	6	7	9	11	29	65		
OLDR	R	V	SS	81	99	101					
OLDS	R	V	SS	80	101						
PROD	R	V	SS	64	75	79	87	89	92	94	
R	ED	V	A A2	1	2	12	13	18	19	39	40
				44	45	50	51	113			
RAD	R	V	SS	62	70	75	81	82	83	86	89
				93	95	97	99	100	101	108	
RL	R	V	SS	82	86	95					
RM	R	V	SS	85	86	97					
RP	ED	V	SS	2	19	25					
RS	ED	V	SS	2	18	23					
R1	R	V	SS	39	42	44	46	49	51	53	57
R2	R	V	SS	40	41	42	45	46	50	52	53
S	R	V	SS	61	67	80	99	101			
SNGL	R	IF		31							
TB	R	V	SS	78	79	83	85	92	94		
TOP	R	V	SS	63	67	79					
W	ED	V	ASA1	1	2	8	9	16	23	25	31
10				14	16						
100				77	93	100	103				
110				79	101	108					
12				12	18						
120				109	112						
130				113	116						
16				21	24	26					
19				17	27						
20				29	31						
30				32	33						
31				35	117						
32				43	46						
34				41	48						
36				50	59						
37				52	57						
38				56	58						
40				47	60	107					
[illegible]				63	102						
5				7	8						
70				69	75						
8				11	28						

```
80                                   66      71      74      76
83                                   86      96      98
90                                   89      91
94                                   92      97
98                                   84      94      99
```

```
C      REAL FUNCTION PA07DD(Z,FZ,N1,A)                                     S/
       DOUBLE PRECISION FUNCTION PA07DD(Z,FZ,N1,A)
C      REAL Z(2),FZ(2),A(N1),P,Q,R,S,T                                     S/
       DOUBLE PRECISION Z(2),FZ(2),A(N1),P,Q,R,S,T
C Z AND FZ CONTAIN COMPLEX NUMBERS.
C A CONTAINS REAL POLYNOMIAL COEFFICIENTS.
C POLYNOMIAL VALUE AT Z IS SET IN FZ AND THE SQUARE OF ITS MODULUS
C IS RETURNED AS FUNCTION VALUE.
       N=N1-1
       T=A(1)
       IF(Z(2).EQ.0.)GO TO 30
       P=Z(1)+Z(1)
       Q=-(Z(1)**2+Z(2)**2)
       R=0.
       IF(N.EQ.1)GO TO 20
       DO 10 K=2,N
       S=R
       R=T
10     T=P*R+Q*S+A(K)
20     FZ(1)=Z(1)*T+Q*R+A(N+1)
       FZ(2)=Z(2)*T
       GO TO 50
30     P=Z(1)
       DO 40 I=1,N
40     T=T*P+A(I+1)
       FZ(1)=T
       FZ(2)=0.
50     PA07DD=FZ(1)**2+FZ(2)**2
       RETURN
       END
```

PROGRAM UNIT PA07DD

ARGUMENTS Z FZ N1 A

NAME	TYPE	USE	ATTRIBUTES	REFERENCES							
A	ED	V	A A1	1	2	4	13	14	19		
FZ	ED	V	ASA1	1	2	14	15	20	21	22	
I	I	V	SS	18	19						
K	I	V	SS	10	13						
N	I	V	SS	3	9	10	14	18			
N1	I	V	A S	1	2	3					
P	ED	V	SS	2	6	13	17	19			
PA07DD	ED	FN	S	1	22						
Q	ED	V	SS	2	7	13	14				
R	ED	V	SS	2	8	11	12	13	14		
S	ED	V	SS	2	11	13					
T	ED	V	SS	2	4	12	13	14	15	19	20
Z	ED	V	A A1	1	2	5	6	7	14	15	17
10				10	13						
20				9	14						
30				5	17						
40				18	19						
50				16	22						

```
   1            SUBROUTINE PA07ED(A,B,Q)
          C     REAL A(2),B(2),Q(2),U,A1,C,D,E                              S/
   2            DOUBLE PRECISION A(2),B(2),Q(2),U,A1,C,D,E
          C IMPLEMENTS THE COMPLEX DIVISION Q=-A/B
          C     U= ABS(B(1))+ ABS(B(2))                                     S/
   3            U=DABS(B(1))+DABS(B(2))
   4            A1=A(1)
   5            C=B(1)/U
   6            D=B(2)/U
   7            E=-(C*C+D*D)*U
   8            Q(1)=(A1*C+A(2)*D)/E
   9            Q(2)=(A(2)*C-A1*D)/E
  10            RETURN
          C                                                                 //
  11            END


PROGRAM UNIT      PA07ED

ARGUMENTS         A      B      Q

NAME     TYPE  USE ATTRIBUTES REFERENCES

A         ED    V     A A1      1     2     4     8     9
A1        ED    V      SS       2     4     8     9
B         ED    V     A A1      1     2     3     5     6
C         ED    V      SS       2     5     7     8     9
D         ED    V      SS       2     6     7     8     9
DABS      ED    IF              3
E         ED    V      SS       2     7     8     9
Q         ED    V     ASA1      1     2     8     9
U         ED    V      SS       2     3     5     6     7
```

References

M.A. Jenkins (1969). Three-stage variable-shift iterations for the solution of polynomial equations with a posteriori error bounds for the zeros. Stanford Report CS 138.

M.A. Jenkins and J.F. Traub (1970). A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. Numer. Math. 14, 252-263.

K. Madsen (1973). A root-finding algorithm based on Newton's method. BIT 13, 71-75.

A.M. Ostrowski (1966). Solution of equations and systems of equations. Academic Press.

G. Peters and J.H. Wilkinson (1971). Practical problems arising in the solution of polynomial equations. J. Inst. Maths. Applics., 8, 16-35.

B.G. Ryder (1973). The FORTRAN verifier: user's guide. Computing science technical report # 12, Bell Telephone Laboratories.

J.H. Wilkinson (1963). Rounding errors in algebraic processes. HMSO.